

Endbericht

der Projektgruppe

Personalisierung internetbasierter Handelsszenarien

DATA **I**N **K**NOWLEDGE **O**UT



Tim Brüggemann, Tina Goldau, Christian Lüpkes, Michael Onken, Matthias Pretzer, Christian Reitenberger, Carsten Saathoff, Helge Saathoff, Ralph Stuber, Insa Stührenberg, Oliver Wien, Guido Zendel, Ralf Krause, Heiko Tapken

Carl von Ossietzky Universität Oldenburg
Fakultät 2 Department für Informatik
Abteilung Informationssysteme
Prof. Dr. Appelrath

© Prof. Dr. Hans-Jürgen Appelrath
Carl v. Ossietzky Universität
Department für Informatik
Escherweg 2
26121 Oldenburg

e-mail: appelrath@informatik.uni-oldenburg.de

erschienen im Jahr 2003

Inhaltsverzeichnis

Teil I

Einleitung

Bei dem vorliegenden Dokument handelt es sich um den Endbericht der Projektgruppe „Personalisierung internetbasierter Handelsszenarien“. Der Endbericht beschreibt Ziele, Vorgehensweise und die Ergebnisse der Arbeit der Projektgruppe und ist in fünf Teile gegliedert:

1. Die *Einleitung* umfasst neben diesem Text eine kurze Beschreibung von Projektgruppen, wie sie laut der Projektgruppenordnung vom 1. Juni 1988 des Departments für Informatik an der Universität Oldenburg durchgeführt werden. Ebenso enthält sie eine Beschreibung der Ausgangslage der Projektgruppenarbeit in Form eines Rahmenkonzepts und einer Anforderungsdefinition des Gesamtszenarios.

2. Die Dokumentation der *Integration* beschreibt den Ablauf des zweiten Teilprojekts der Projektgruppe. Kernelemente dieses Kapitels sind die Anforderungsdefinition der Integration in Form einer objektorientierten Analyse sowie der objektorientierte Entwurf.

3. Der als *Analyse* bezeichnete Teil des Endberichts umfasst die Beschreibung der Anforderungen, die an das von der Projektgruppe entwickelte Personalisierungs-Framework sowie an die Data-Mining-Bibliothek gestellt werden. Des Weiteren sind der Entwurf der entwickelten Komponenten sowie technische Handbücher enthalten.

4. Im *Personalisierungskonzept* werden Anwendungsmöglichkeiten für die entwickelte Software im Rahmen der Projektgruppenarbeit untersucht. Es wird sowohl die Möglichkeit der Verwendung temporaler als auch nicht-temporaler Analyseverfahren betrachtet.

5. *Ergebnisse und Schlussbetrachtungen* der Projektgruppe werden in Teil V des Dokuments zusammengefasst. Es wird insbesondere ein für die Präsentation von Analyseergebnissen entwickelter, auf dem Personalisierungskonzept basierender Online-Shop beschrieben. Außerdem bietet der Teil einen Soll-Ist-Vergleich zwischen den geforderten und den tatsächlich erbrachten Leistungen sowie eine Zusammenfassung der Projektgruppenergebnisse und einen Ausblick im Hinblick auf zukünftige Entwicklungen.

Begleitende Texte, wie z.B. die für die Integration verwendeten Übersetzungsmatrizen, befinden sich im *Anhang* des Endberichts.



1 Rahmenbedingungen

Im Folgenden wird beschrieben, was unter einer Projektgruppe zu verstehen ist. Ein Dokument zu dieser Thematik stand bereits zum Jahreswechsel 87/88 im wesentlichen fest und wurde nach kleinen Änderungen am 1.6.1988 von der Studienkommission des Fachbereichs Informatik der Universität Oldenburg verabschiedet [Inf88].

1.1 Definition von Projektgruppen

Projektgruppen sollen die Besonderheiten eines Fortgeschrittenenpraktikums, eines Seminars und einer Studienarbeit vereinen und zugleich berufstypische Arbeitsweisen im Studium vermitteln. Eine Projektgruppe besteht aus acht bis zwölf Studierenden, die ein Jahr lang ein umfangreiches Problem bearbeiten. Insgesamt wird diese Lehrveranstaltung mit 15 Semesterwochenstunden angesetzt. Die Themen für Projektgruppen können sowohl aus der *Kern-Informatik* als auch aus ihren Anwendungsgebieten stammen. Es ist möglich (und auch sinnvoll), dass sich an einer Projektgruppe Studierende anderer Fachrichtungen oder externe Kooperationspartner beteiligen. Eine Projektgruppe sollte i. A. durch eine Stammvorlesung und/oder eine Spezialvorlesung vorbereitet werden. In der Regel werden aus einer Projektgruppe mehrere Diplomarbeiten hervorgehen.

1.2 Erläuterungen zum Zweck von Projektgruppen

Zu vermittelnde berufstypische Arbeitsweisen sind:

- Arbeiten im Team (insbesondere: Präzisierung und Definition von Schnittstellen, Aufgabenverteilung, Zuständigkeiten und Verläßlichkeiten in einer Gruppe)

- Umfassendere Betrachtungsweisen bei der Software-Entwicklung (Kennenlernen des gesamten Software-Lebenszyklusses, Verwen-

dung unterschiedlicher Spezifikationssprachen, Soll- und Ist-Analysen, Kosten-Nutzen-Analysen, Einsatz- und Auswirkungsproblematik, Standard- und kundenspezifische Software)

- Einsatz von Werkzeugen (Sichtung vorhandener Software, Planung und Auswahl von Sprachen, Nutzung von Software-Entwicklungswerkzeugen, maschinenabhängige und maschinenunabhängige Konzepte)
- Konkrete Erstellung von Software unter gleichzeitiger Anfertigung einer Dokumentation und weiterer Materialien (Handbücher, Konfigurationen, Wartungsanweisungen usw.)
- Arbeitsstil und persönliche Befähigungen (Organisation umfangreicher Projekte, Präsentation von Resultaten und Teilnahme an Diskussionen, Sitzungen mit präziser Protokollierung, Planungs- und Konfliktmanagement, Einarbeitung und Beschaffung von Literatur, Einblick in arbeitspsychologische Phänomene)

1.3 Hinweise für Veranstalter und Studierende

Entsprechend dieser Ziele sollte eine Projektgruppe nach folgendem Schema geplant werden und ablaufen:

1. Die Vorgaben durch die Veranstalter umfassen:
   - Grobziele der Projektgruppe mit Themenstellung und Zielsetzungen
   - Zeitplanung über zwei Semester
   - Literaturangaben und Vorträge für die Seminarphase
   - Vorschläge für Selbstverwaltungsaufgaben
   - Teilnahmevoraussetzungen
   - präzise Kriterien für die Erlangung des Projektgruppenscheins
   - benötigte Ressourcen des Fachbereichs und Entwicklungsumgebungen
2. Minimalergebnisse: Von jedem Teilnehmer wird die aktive Mitarbeit an den Projektgruppensitzungen, die Vorbereitung und

Abhaltung eines oder zweier Seminarvorträge, die Erfüllung von bestimmten Verwaltungsaufgaben innerhalb der Projektgruppe, die Mitarbeit an einem Zwischen- und einem Endbericht und die Erfüllung der übertragenen Programmieraufgaben erwartet. Frühzeitige Aussprachen über mangelnde Mitarbeit oder Desinteresse bei Beteiligten gehören zum Konfliktmanagement und sollten keinesfalls verdrängt werden. Von besonderer Bedeutung ist die kontinuierliche Erarbeitung einer Dokumentation, an der alle Teilnehmer in gleichem Maße mitwirken müssen. Dabei geht es nicht um ein Handbuch (das ebenfalls erstellt werden muss), sondern um die systematische Darstellung z.B. folgender Bereiche:

- Problemstellung, Literatur, Überblick
- Ist-Analyse und Soll-Konzept
- Anforderungen, Spezifikationen, Benutzerschnittstellen
- Empirische und formale Evaluation
- Modulbeschreibungen und Implementierung
- Integration und Tests
- Wartung, Fehlerfälle, Portierungsmöglichkeiten
- Erweiterungen, Ausblick.

3. Die Zeitplanung kann nach folgendem Schema ablaufen:

   - Vorlesungsfreie Zeit vor der Projektgruppe:
     Vorbereitung der Vorträge der 1. Seminarphase, Einarbeitung in die Entwicklungsumgebungen
   - Erstes Semester: Seminarphase (ca. vier Wochen), Planungs- und Entwurfsphase (ca. vier Wochen), Spezifikationen, Aufgabenverteilung, Probeimplementierungen (ca. vier Wochen)
   - Vorlesungsfreie Zeit:
     Beginn der Implementierungsphase
   - Zweites Semester: Weiterführung der Implementierungsphase, Integrationsphase und Tests (ca. vier Wochen), Seminarphase, Erprobung, Handbuch, Dokumentation
   - Vorlesungsfreie Zeit nach der Projektgruppe: Präsentation der Projektgruppenergebnisse im Fachbereich Informatik.

Besuche bei anderen Fachbereichen und/ oder Anwendern sollten eingeplant werden; empfehlenswert ist eine Exkursion zu Firmen/ Institutionen, die sich mit ähnlichen Fragen wie die Projektgruppe beschäftigen.

4. Selbstverwaltungsaufgaben in der Projektgruppe: Erfahrungsgemäß fallen folgende „Ämter“ an:
    - Meilensteinüberwachung/ Projektplaner
    - Schnittstellenüberwachung und -präzisierung
    - Tester
    - Archivar
    - Bibliothekar und Literaturbeschaffer
    - Experten für spezielle Programmiersprachen
    - Experten für spezielle Rechnersysteme
    - Experten für spezielle Softwaresysteme
    - Kümmerer und Haushaltsüberwacher
    - Webbeauftragter

Hinweis: Projektgruppen sind von den Gremien des Fachbereichs Informatik zu genehmigen. Hierzu ist zunächst das Thema und die Ankündigung (einschl. der unter 1 bis 3 genannten Punkte) im vorangehenden Semester der Studienkommission zur Bestätigung vorzulegen. Da eine Projektgruppe ein Seminar, ein Fortgeschrittenenpraktikum und eine Studienarbeit ersetzt, muss der Diplom-Prüfungsausschuss die Äquivalenz der Projektgruppe mit diesen Studienleistungen genehmigen. Anschließend kann die Projektgruppe ausgeschrieben werden. Eine Projektgruppe soll nicht begonnen werden, wenn weniger als acht Studierende an ihr teilnehmen.

1.4 Projektgruppenantrag

Die nachfolgende Übersicht faßt den im Fachbereich gestellten Antrag des Veranstalters auf Durchführung der Projektgruppe in seinen formalen Angaben und der Aufgabenstellung zusammen.

1.4.1 Formalia

Im Folgenden werden allgemeine Angaben über die Projektgruppe gegeben.

Veranstalter

Prof. Dr. H.-J. Appelrath
Informatik-Assistent Ralf Krause
Dipl.-Informatiker Heiko Tapken

Zeitraum

WS 02/03 und SS 03

Umfang

Bei der Planung ist eine Basierung der Abschätzung auf der Grundlage von Personen-Tagen üblich. Ein Personen-Tag umfasst acht Stunden. Für die Projektgruppe sind folgende Aufwände zu verplanen (ein Jahr - zwei Wochen Urlaub - vier Wochen Seminar):

- Fachbereich 10 (Informatik): 7 Personen * 20 Stunden * 46 Wochen= 6.440 Personen-Stunden
- Fachbereich 4 (Wirtschaftswissenschaften): 5 Personen * 15 Stunden * 46 Wochen= 3.450 Personen-Stunden
- Insgesamt 9.890 Personen-Stunden/8 Stunden, d.h. 1.236 Personen-Tage

Lehrveranstaltungsäquivalent

1 Seminar
1 Fortgeschrittenenpraktikum
1 Studienarbeit

Inanspruchnahme von Fachbereichsressourcen

Der Rechner- und Softwarebedarf wird durch Ressourcen der veranstaltenden Abteilung befriedigt. Ein Raum für Sitzungen steht im OFFIS-Gebäude zur Verfügung.

Teilnahmevoraussetzungen

Abgeschlossenes Grundstudium mit erfolgreich abgeschlossenem Vordiplom zu Beginn des WS 02/03.

1.4.2 Aufgabenstellung

Das Ziel dieser Projektgruppe ist es, ein System zu konzipieren und zu realisieren, das personalisierte, internetbasierte Handelsszenarien abbildet.

Die Teilnehmer der Projektgruppe sollen sich in die Lage eines Kartenanbieters (analog zu Payback, Miles&More) versetzen. Es soll ein neues Szenario für die Nutzung von Kundenkarten realisiert werden, das von mehreren Händlern genutzt wird, um dieses später weiteren Händlern „zu verkaufen“.

Kartensysteme

Kartensysteme bieten Vorteile für Händler, Anbieter von Kartensystemen sowie für Kunden. Kunden haben die Möglichkeit, ein personalisiertes Informations- und Warenangebot in Anspruch zu nehmen. Dieses wird dadurch ermöglicht, dass Handelstransaktionen von partizipierenden Kunden von den Händlern an den entsprechenden Kartenanbieter übermittelt werden. Dafür erhält der Händler vom Kartenanbieter ein Entgelt. Der Kartenanbieter analysiert die Kundendaten. Heute beschränken sich die von den Kartenanbietern durchgeführten Analysen i.d.R. auf die Segmentierung von Kunden in Kundengruppen (Bsp. Amazon). Händler sind bestrebt, ihren Kunden ein möglichst gutes Angebot zu machen. Dies können sie gewährleisten, indem sie von Kartenanbietern Analyseergebnisse in Form von Kundengruppenbeschreibungen und Einkaufsregelbeschreibungen einkaufen. Es kann z.B. unter Verwendung eines Publish/Subscribe Ansatzes erfolgen. Die gekauften Informationen kann der Händler nutzen, um Kunden kundengruppengerechte Angebote z.B. über das Internet unterbreiten zu können.
Die sich ergebenden Vorteile für alle partizipierenden Gruppen werden im Folgenden dargestellt: Kunden erhalten ein auf sie zugeschnittenes Informations- und Warenangebot; der Händler kann seine Kunden besser an sich binden und der Kartenanbieter profitiert durch den Verkauf seiner Analyseergebnisse. Tatsächlich profitieren jedoch in der Regel nur die Kartenanbieter und die Händler, da die Kunden ihre Daten für eine geringe Gegenleistung bereitstellen. Der Kunde wird durch die Weitergabe seiner Daten zum „gläsernen Kunden“, wodurch Probleme entstehen können (z.B. bezüglich des Datenschut-

zes). Der Händler hingegen begibt sich durch die Einführung von Kartensystemen in eine Abhängigkeit zu dem mit ihm kooperierenden Kartenanbieter. Für Kunden und Händler kann es ferner von Nachteil sein, dass ihre Daten in einem Informationsverbund gespeichert werden, auf den sie nur beschränkt Einfluss nehmen können. Wenn z.B. ein Kartenanbieter einer Branche mit einem Händler kooperiert, sollte er z.B. (auch technisch) daran gehindert werden, diese Daten einem Konkurrenten dieses Händlers zukommen zu lassen. Nachteile für den Kartenanbieter ergeben sich i.d.R. nicht. Die Projektgruppe versetzt sich in die Rolle eines Kartenanbieters.
Um ein System entwickeln zu können, dass von einem Kartenanbieter angeboten werden kann, ist es notwendig, sowohl die Seite des Kartenanbieters als auch die Rolle des Händlers zu betrachten:

- Händler
  Ein Händler partizipiert in der zu betrachtenden Architektur in zweierlei Formen von Handelsbeziehungen. Er handelt einerseits mit Kunden durch den Verkauf von Gütern. Andererseits verkauft er dem Kartenanbieter Informationen und kauft Analyseergebnisse von diesem. Er sammelt und speichert alle Daten (Verkaufsdaten, Interessenten, Web-logs, etc.), die mit seinem Geschäft zu tun haben. Einen kleinen Teil dieser Daten (die Daten, die mit der angebotenen Karte zu tun haben, i.d.R. die Informationen des Kassenbons) übermittelt er an den Kartenanbieter. Ihm stehen jedoch weit mehr Informationen zur Verfügung, die er zur Individualisierung seines Angebotes nutzen könnte. Hierbei könnte er z.T. auf die vom Kartenanbieter gelieferten Analyseergebnisse zurückgreifen. Kennt er die Präferenzen seines Kunden als Individuum (Tante-Emma-Laden--Metapher), so kann er z.B. in einem elektronischem Shopsystem eine personalisierte Präsentation seiner Waren durchführen.

- Kartenanbieter
  Der Kartenanbieter geht Handelsszenarien mit Händlern und Endkunden ein. Um an die Daten zu kommen, die für die Identifikation von Kundengruppen benötigt werden, „bezahlen“ die Kartenanbieter die Kunden in Form von Prämien (Punktesysteme, Prämien, o.ä.). An die Daten von Kunden kommen die Kartenanbieter durch die Händler, die ein Entgelt (z.B. in Form von Analyseergebnissen) für das Processing (Bearbeiten) der Daten

erhalten. Der Kartenanbieter sammelt, integriert, bereinigt und speichert die Daten dauerhaft. Anhand dieser aufbereiteten Daten ist der Kartenanbieter in der Lage, Analysen verschiedener Art durchzuführen. Hierzu zählen die Warenkorbanalyse, die Kundengruppierung, die Konzeptbeschreibung sowie die Herleitung von Regeln. Ein fiktives Beispiel für eine solche Regel ist: Ein junger Vater Ende 20, der in einem Supermarkt Pampers kauft, kauft mit einer Wahrscheinlichkeit von x% auch einen 6er Träger Bier. Die Ergebnisse dieser Analysen kann er Händlern zum Kauf anbieten. Die an diesem Handel teilnehmenden Personen können andere sein als die, von denen die Daten gesammelt und ausgewertet wurden.

Zielsetzung

Die Aufgaben der Projektgruppe gliedern sich in zwei Aufgabengebiete:

1. Herleitung personalisierter Daten
    - Kundensegmentierung: Wie können (aggregierte) Kundendaten mehrerer Händler genutzt werden, um Kunden in Gruppen einzuteilen? (Kartenanbieter)
    - Individualisierung: Wie können aus den Daten individuelle, interessante Angebote für den Kunden erstellt werden? (Händler)
2. Internetbasierte, individualisierte Präsentation von Daten
    - Wie können Händler individuelle Angebote unterbreiten? (Der Kunde soll hierbei personalisierte Angebote erhalten.)
    - Für einen Händler soll exemplarisch ein Shopsystem konzipiert und realisiert werden.

Entwicklungsumgebung

- Hardware:
    - SUN-Workstations
    - DEC-Workstations

  - PCs unter Windows 2000/XP und Linux

- Software:
  - Mozilla Navigator
  - Internet Explorer
  - Java 1.4 SDK
  - Eclipse IDE
  - CVS (Versionskontrolle)
  - Perl (zur Realisierung von CGI-Skripten, soweit notwendig)
  - LaTeX (für die Dokumentation)
  - JUnit
  - Berkeley DB

Minimalergebnisse, Scheinkriterien

- Minimalergebnisse müssen erreicht werden (siehe Abschnitt 1.3)
- Aktive Mitarbeit bei der Analyse, Konzeption und Implementierung
- Erfüllung übertragener Aufgaben
- Präsentation und Ausarbeitung von Seminarvorträgen
- Erstellung von Zwischen- und Endbericht sowie anfallenden Dokumentationen

Meilensteinplanung

Die Meilensteinplanung des Projektes ist dem Kapitel 21.9 zu entnehmen.

1.4.3 Literatur

- http://www-is.informatik.uni-oldenburg.de/lehre/lehre.html
- http://www.diko-project.de

1.5 Teilnehmer

Die Projektgruppe setzt sich aus folgenden zwölf Studierenden zusammen:

- Tim Brüggemann
- Tina Goldau
- Christian Lüpkes
- Michael Onken
- Matthias Pretzer
- Christian Reitenberger
- Carsten Saathoff
- Helge Saathoff
- Ralph Stuber
- Insa Stührenberg
- Oliver Wien
- Guido Zendel

Veranstalter sind Prof. Dr. H.-J. Appelrath, Informatik-Assistent Ralf Krause und Dipl.-Inform. Heiko Tapken.

2 Anforderungsdefinition Gesamtszenario

Die Datenmengen in Organisationen sind im Laufe der Zeit und speziell in der heutigen modernen, von Computern geprägten Gesellschaft enorm angestiegen. Um aus diesen Daten neues Wissen generieren zu können, müssen die Informationen regelmäßig gesammelt und aufbereitet werden. Händler wollen in der heutigen Zeit ihren Kunden ein möglichst gutes, auf sie zugeschnittenes Angebot unterbreiten, um ihre Umsätze zu erhöhen und um Kunden zu binden. Eine Bindung von bereits bestehenden Kunden ist kostengünstiger als eine Gewinnung von Neukunden. Für die Kundenbindung ist es notwendig, auf die Wünsche und Interessen der vorhandenen Kunden und potentiellen Neukunden möglichst individuell einzugehen. Eine Grundlage für personalisierte Angebote ist z.B. der Einsatz eines Kartensystems. Das Kartensystem ermöglicht es, Kunden zu erkennen und damit Einkäufe einem Kunden zuzuordnen. Aus diesem Wissen können dann Analyseergebnisse erstellt werden, die das Kundenverhalten für Händler transparenter machen. Aus Händlersicht wird der Kunde im optimalen Fall zum „gläsernen Kunden“. Aus den genannten Gründen ist Personalisierung zu einer wichtigen Marketinggrundlage geworden.

Vor diesem Hintergrund wurde der Projektgruppe „Personalisierung internetbasierter Handelsszenarien“ die Aufgabe gestellt, ein System zu entwickeln, welches personalisierte, internetbasierte Handelsszenarien abbildet (vgl. Abbildung 2.1). Es soll eine Java-Bibliothek entwickelt werden, die Muster in Daten findet, welche als Grundlage eines Personalisierungskonzeptes genutzt werden können. Zudem soll ein exemplarischer Online-Shop entwickelt werden, der die Ergebnisse der Bibliothek visualisieren und somit die Bibliothek auf ihre Funktion hin testen soll.

Die Händler sind beim Einsatz eines Kartensystems bestrebt, für dieses zu werben und möglichst viele Kunden für den Gebrauch der Karte zu motivieren. Da sich der Kunde bei einer Bestellung oder bei

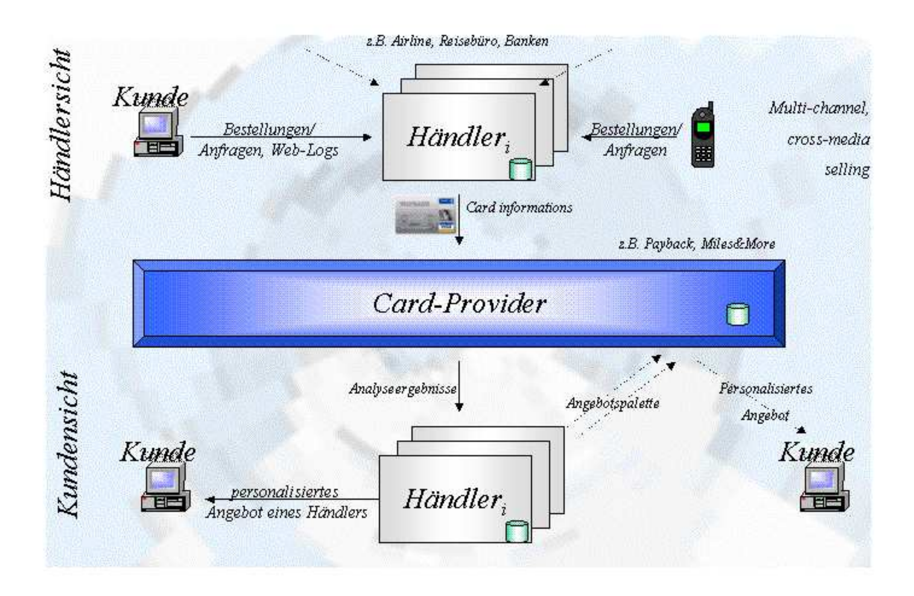


Abbildung 2.1: Aufgabenstellung

einem Kauf anhand seiner Karte bzw. seiner Kartennummer identifiziert, kann der Händler die erhaltenen Daten personenbezogen zuordnen. Im Internet können Kundenanfragen und das Verhalten direkt mit dem entsprechendem Interessenten in Verbindung gebracht werden. Das Kaufverhalten wird dadurch für die Händler auswertbar.
Um aus den gesammelten Kunden- und Verkaufsdaten Analyseergebnisse zu gewinnen, werden die Daten von den Händlern zum Anbieter des Systems transferiert. Damit die Kunden durch den Gebrauch der Karte mit der Verwendung (im Regelfall nur innerhalb des Unternehmens) ihrer persönlichen Daten einverstanden sind, erhalten sie eine Belohnung beispielsweise in Form von Prämien. Da der Kartenanbieter die Möglichkeit hat, die Daten für sämtliche Analysen in seinem System (also auch für andere Händleranalysen) nutzen zu können, erhalten die Händler wiederum für die Vorverarbeitung der Daten ein Entgelt. Der Kartenanbieter ist in der Lage, aus den Daten neues Wissen für den Händler zu generieren. Die Händler wiederum bezahlen den Kartenanbieter für die Analyseergebnisse und die Bereitstellung

des Systems. Da diese Analysen zeitaufwändig und komplex sind, müssen die Händler diese Leistung vom Kartenanbieter erkaufen, da sie u. U. nicht in der Lage sind, diese Ergebnisse selbst zu generieren. Aus diesen Erkenntnissen können die Kunden individuell beworben und damit die Marketingstrategien der Händler optimiert werden. Die Bezahlfunktion wird innerhalb der Projektgruppe vernachlässigt, da die Systemimplementierung im Vordergrund steht und die Eingliederung einer Bezahlkomponente zu einem späteren Zeitpunkt problemlos erfolgen kann.
Basierend auf dieser Aufgabenstellung hat sich die Projektgruppe entschieden, das im Folgenden erklärte Szenario umzusetzen (vgl. Abbildung 2.2). Der Schwerpunkt liegt dabei auf temporalen Aspekten bei der Datenhaltung, den Analysen und somit ebenfalls in den Analyseergebnissen.

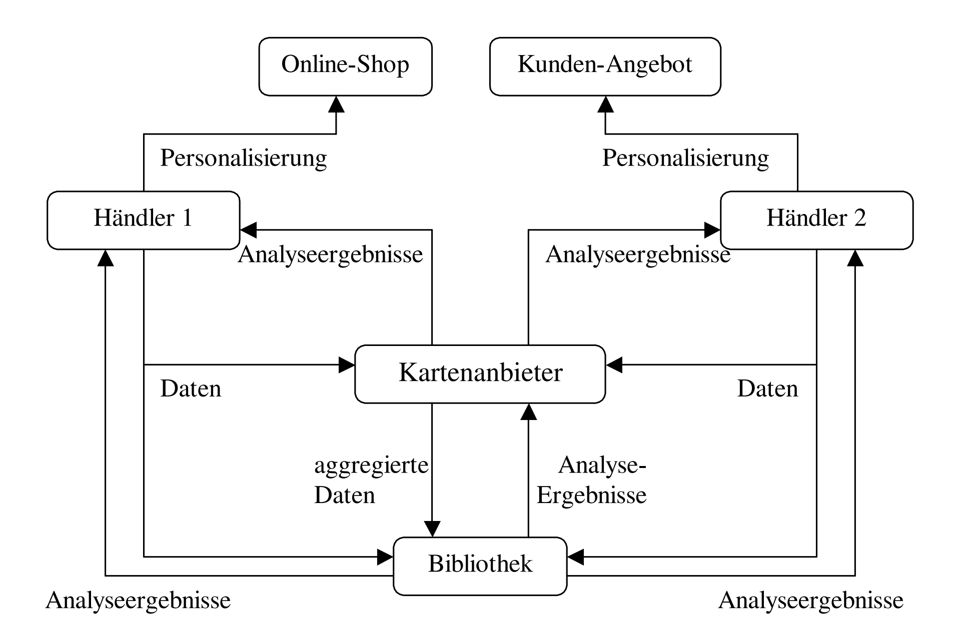


Abbildung 2.2: Gesamtszenario

Die Händler stellen einen Bestandteil des Gesamtszenarios dar. Die Schnittstelle der Händler zum Kunden liegt in der Präsentation eines individuellen Angebotes und in dem Einsatz eines personalisierten Online-Shops. Die Händler sind in Online- und Offline-Händler

unterteilt, um ein umfassendes Szenario abzubilden. Online-Händler besitzen einen Online-Shop (Händler 1), während Offline-Händler für ihre Warenpräsentation und ihren Warenverkauf Filialen nutzen (Händler 2). Die Online-Händler erlangen die Verkaufs- und dazugehörigen Kundendaten durch die Registrierung der Kunden im Internet sowie durch die dort anfallenden Transaktionen und Analyse der Weblogs. Die Kundendaten bei den Offline-Händlern werden entweder durch Schätzungen (Alter, Geschlecht) und Befragungen an der Kasse (PLZ) oder über Kundenkartenanträge aufgenommen. Die dazugehörigen Verkaufsdaten werden über die Bons erfasst. Das Gesamtszenario der Projektgruppe beschränkt sich auf nur einen Online- und einen Offline-Händler, ist jedoch beliebig erweiterbar.
Des Weiteren wird ein Kartenanbieter berücksichtigt, der Kartensysteme anbietet und Analyseergebnisse bereitstellt. Es besteht die Möglichkeit, weitere Kartenanbieter aufzunehmen.
Der Kartenanbieter integriert und speichert die Kunden-, Transaktions- und Produktdaten der Händler. Diese Daten sind Basis für die Analysen, die innerhalb der Klassenbibliothek WEKA und deren Erweiterungen ablaufen. Die daraus resultierenden Analyseergebnisse werden von den Händlern gekauft und an diese übertragen. Die Online-Händler nutzen die Ergebnisse für ihren Online-Shop, während die Offline-Händler daraus personalisierte Angebote in ihrem Filialsystem erstellen. Zusätzlich können die Händler eigene Analysen mit WEKA und deren Erweiterung auf ihren Datenbanken durchführen.

2.1 Systemeinsatz und Systemumgebung

Für den Datenbankentwurf der Händler- und Kartenanbieterdatenbanken, der als Grundlage für die weitere Projektarbeit dient, wird das Windows-Programm ERwin verwendet. Als Datenbank wird Oracle (Version 9i) zugrunde gelegt. Diese Datenbank läuft auf den von der Projektgruppe genutzten Rechnern unter Solaris. Als Programmiersprache wird Java in der Version 1.4 benutzt. Das Programm Eclipse wird als Arbeitsumgebung für die Programmierung verwendet. Die Implementierung erfolgt in Java, um eine möglichst hohe Plattformunabhängigkeit erreichen zu können. Als Grundlage für die Personalisierungsbibliothek dient das freie Softwareprojekt WEKA. Der Online-Shop basiert auf Analyseergebnissen des Kartenanbieters.

Es wird darauf verzichtet, ein existierendes Shopsystem anzupassen. Stattdessen wird ein eigener exemplarischer Online-Shop implementiert.

2.2 Funktionale Anforderungen

Im Folgenden werden die funktionalen Anforderungen an das Gesamtszenario erläutert.
Grundlage für das dargestellte Gesamtszenario der Projektgruppe ist der Entwurf der Datenbankschemata des On- sowie Offline- Händlers und das Schema des Kartenanbieters. Diese Schemata werden im Anschluss ausführlich getestet und dokumentiert. Um die Schemata testen zu können, werden Testdaten, basierend auf Zufallswerten durch Perl-Skripte erzeugt und z.T. manuell eingegeben. Transaktionen mit kalendarischen Mustern werden hierbei mit dem Algorithmus tBasket[LNWJ] generiert. Nach erfolgreichem Testen werden die Testdaten in den Schemata gelöscht.
Die Schemata der Händler werden von den Händlern mit den vorhandenen Kunden-, Transaktions- und Produktdaten gefüllt. Das Kartenanbieterschema beinhaltet initial keine Daten.
Nach dem Entwurf der DB-Schemata ist die Projektarbeit in die zwei Teilaufgaben Integration und Analyse unterteilt. Die *Integration* hat im Rahmen des Gesamtszenarios die Aufgabe, die Daten der Händler aus den jeweiligen Datenbanken in die Datenbank des Kartenanbieters zu integrieren. Der Vorgang der Integration, der die Daten lädt, kann auf verschiedene Weise aufgerufen werden. Es gibt die Möglichkeit, der Software mittels Parameteterübergabe mitzuteilen, ob sie

- für das Kartenanbieterschema essentielle Initialimportdaten in die Kartenanbieterdatenbank einfügen soll, um sie so auf die weiteren Importe vorzubereiten,
- die Daten der Händlerdatenbank des Händlers 1 in die Kartenanbieterdatenbank integrieren oder
- die Daten der Händlerdatenbank des Händlers 2 in die Kartenanbieterdatenbank integrieren soll.

Die Integration hat hinsichtlich der Vorverarbeitung der Daten (Preprocessing) lediglich die Aufgabe, ggf. fehlende Attribute, die jedoch

temporal	**nicht temporal**
Temporale Assoziationsanalyse	Clusteranalyse
Sequentielle Musteranalyse	Klassifikation
Verläufe temporaler und nicht-temporaler Muster	Assoziationsanalyse

Tabelle 2.1: Auswahl Analyseverfahren

in Primärschlüsseln im Kartenanbietersschema vorausgesetzt werden, zu ergänzen.
Eine nähere Beschreibung der speziellen Funktionalität lässt sich der Anforderungsdefinition der Integration entnehmen. Dort liegt ebenso eine komplette syntaktische Beschreibung der möglichen Parameterübergaben vor.
Die *Analyse* hat die Aufgabe sich mit der Auswertung der integrierten Händlerdaten in der Kartenanbieter-Datenbank zu beschäftigen. Die Basis der Datenanalyse stellt der KDD-Prozess dar. Dieser Prozess gliedert sich in die Phasen Selektion, Vorverabeitung, Transformation, Data Mining und Interpretation. Es stehen mehrere Verfahren zum Data Mining zur Auswahl (siehe Tabelle 2.1), mit denen die Händlerdaten analysiert werden können.

Die Projektgruppe hat sich aus Zeitgründen dafür entschieden, sich auf ein nicht-temporales (Clustering) und ein temporales (kalendarische Muster) Verfahren zu beschränken. Durch Clustering können Daten in Gruppen möglichst hoher Ähnlichkeit eingeteilt werden. Als Arbeitsbasis wird die Klassenbibliothek WEKA verwendet. Zur Umsetzung des Cluster-Verfahrens wird WEKA genutzt, da dort dieses Verfahren bereits vorhanden ist. Für die temporalen Assoziationsregeln wird die Bibliothek um temporale Aspekte erweitert. So sollen Cluster entdeckt und kalendarische Assoziationsregeln gefunden werden können. Es wird zu den vorhandenen Clusteralgorithmen von WEKA ein eigener Algorithmus implementiert, der die kalendarische Assoziationsanalyse in WEKA ermöglicht. Die Bibliothek soll hierbei metadatengesteuert sein. Es soll eine Portabitität des Systems gewährleistet werden, indem das Framework überall nutzbar sein soll. Das Framework wird in Java programmiert und dient als Schnittstelle zu WEKA. Es ruft die Analysealgorithmen in WEKA auf und speichert die Ergebnisse in einem Austauschformat. Dieses liegt als Datei

oder Datenbanktabelle vor und kann für weitere Analysen verwendet werden. Eine nähere Beschreibung der speziellen Funktionalität lässt sich der Anforderungsdefinition der Analyse entnehmen.
Zur Überprüfung der Ergebnisse der Personalisierung wird auf Basis der Datenbank des Online-Händlers sowie den Analyseergebnissen (Clustering) ein Online-Shop erstellt. Dort werden auch die temporalen Ergebnisse der kalendarischen Mustererkennung als Präsentation umgesetzt.
Es wird angestrebt, eine baumartige Katalogstruktur darzustellen. Zudem soll der Online-Shop Produktdetailseiten und Warenkörbe enthalten sowie die Möglichkeit der Online-Bestellung bieten. Hinsichtlich der Personalisierung soll auf den Detailseiten eines Produktes andere Produkte gemäß den Assoziationsregeln für den Kunden eingeblendet werden. Ferner soll guten Kunden Sonderangebote unterbreitet werden. Außerdem sollen Produkte innerhalb der Katalogstruktur gemäß den Vorlieben und Interessen des jeweiligen Kunden besonders betont werden. Konkretere Designentscheidungen sowie Inhalt des Online-Shops werden zum Zeitpunkt der Realisierung getroffen (vgl. Meilensteinplan Abbildung 2.3).

Die Abbildung 2.3 zeigt eine Übersicht in Form eines Meilensteinplanes über den Zeitrahmen, in dem die einzelnen Teilprojekte zu realisieren sind.

Nr	Meilenstein	Start	Ende	Dauer
1	Fertigstellung der Ausarbeitungen	01.10.2002	31.12.2002	13,14w
2	Entwurf DB-Schemata	01.11.2002	07.12.2002	5,29w
3	Test u. Dokumentation DB-Schemata	08.12.2002	31.01.2003	7,86w
4	Erstellung Zwischenbericht	01.02.2003	31.03.2003	8,43w
5	Fertigstellung Zwischenbericht	01.04.2003	30.04.2003	4,29w
6	Anforderungsdefinitionen Gesamtszenario, Integration, Analyse	31.01.2003	03.03.2003	4,57w
7	Fertigstellung Integration	10.03.2003	04.05.2003	8w
8	Fertigstellung Analyse	17.03.2003	01.07.2003	15,29w
9	Erstellung zweiter Analyse-Algorithmus	02.07.2003	18.08.2003	6,86w
10	Konzeption und Anforderungsdefinition Shop	09.06.2003	17.08.2003	10w
11	Fertigstellung Shop	18.08.2003	29.09.2003	6,14w
12	Fertigstellung Endbericht	17.08.2003	30.09.2003	6,43w

Abbildung 2.3: Meilensteine

Nach der Erläuterung der Integration, Analyse und des Shops werden im Folgenden die Beziehungen zwischen diesen einzelnen Komponenten beschrieben. Die Integration und die Analyse sind unabhängig voneinander, da die Händler ebenso selber die Möglichkeit haben sollen, auf die Bibliothek zuzugreifen. Die Integration wird in einem feststehendem Rhytmus (z.B. jede Nacht) von außerhalb aufgerufen, so dass die Kartenanbieterdatenbank für geplante Analysen ausreichend aktuell ist. Die Integration wird damit vom Kartenanbieter gesteuert. Die Analyse wird vom Framework angestoßen. Der Online-Shop dient der Überprüfung und Visualisierung der Ergebnisse und folgt damit als letzter Schritt nach der Analyse.

2.3 Nicht-funktionale Anforderungen

Für das System existieren folgende nicht-funktionale Anforderungen:

- Portabilität: Durch den Einsatz von Java ist das System plattformunabhängig einsetzbar, wodurch eine hohe Flexibilität gegeben ist.
- Stabilität: Fehlerbereinigung im Vorfeld wird durch das System gewährleistet. Dadurch ist das System stabil und dementsprechend fehlerresistent.
- Skalierbarkeit: Das System gewährleistet seine Funktionsfähigkeit auch bei großen Datenmengen.
- Benutzerfreundlichkeit (Übergabe von Steuerungsparameter): Das System bietet eine einfache Bedienung und Übersichtlichkeit.
- Wartbarkeit: Das System ist ausführlich dokumentiert und besitzt eine übersichtliche Programmstruktur, die die Wartung erleichtert.
- Sicherheit: Der Zugriff auf die Datenbanken wird nur für berechtigte Nutzer erlaubt, wodurch eine Sicherheit der Daten gewährleistet wird. Die Verbindungen sind hingegen nicht gesichert, da dieses nicht der Aufgabenstellung zu entnehmen ist und eines hohen Arbeitsaufwandes bedarf.

Eine wichtige Qualitätsanforderung ist eine detaillierte Dokumentation des Systems.

2.4 Benutzerschnittstellen und Fehlerverhalten

Das Programm kann Fehler abfangen und verursacht keine Systeminstabilität. Stattdessen wird eine Fehlermeldung ausgegeben. Eine nähere Beschreibung des Fehlerverhaltens sowie der Benutzerschnittstellen lässt sich den Anforderungsdefinitionen der Integration (siehe Abschnitt 4.1) und Analyse (siehe Kapitel 10) entnehmen.

2.5 Dokumentationsanforderungen

Die Programmdokumentation beinhaltet die Anforderungsdefinitionen, die Entwürfe bestehend aus der objektorientierten Analyse(OOA) und des objektorientierten Designs (OOD), die dokumentierten Source-Codes sowie Benutzer- und Entwickler-Handbuch. Klassen, Attribute und Methoden werden mit Hilfe von Javadoc dokumentiert. Sämtliche anderen notwendigen Dokumentationen werden in LaTeX erstellt. Es werden die Editoren WinEdt, Xemacs, vim und Nedit verwendet. Die Dokumentation in Javadoc erfolgt in englischer Sprache.

2.6 Abnahmekriterien

Die genannten funktionalen Anforderungen sind zu erfüllen, während die nichtfunktionale Anforderungen möglichst optimal realisiert werden sollen. Die aus der Analyse entstandenen Cluster werden in einem Online-Shop evtl. zur Realisierung eines Recommender-Systems[1] herangezogen.

[1] Recommender Systeme sind „Empfehlungssysteme“. Sie schließen automatisch von vorhandenen Informationen auch auf neue Daten

Teil II

Integration

3 Implementierung des Integrationsszenarios

In diesem Abschnitt wird das Vorgehen der Integrationsgruppe anhand des Wasserfallmodelles beschrieben. Anschließend werden die einzelnen Phasen genauer betrachtet.

3.1 Vorgehensweise in Anlehnung an das Wasserfallmodell

Die Vorgehensweise der Teilgruppe Integration lässt sich in mehrere, voneinander getrennte Aktivitäten gliedern. Jede Aktivität beginnt und endet mit einem klar definierten Ereignis. Die Aktivitäten stimmen dabei weitestgehend mit den Phasen des Wasserfallmodelles, einem Vorgehensmodell zur Softwareentwicklung, überein.

Nach Vorgehensweise des Wasserfallmodelles wird die Durchführung der einzelnen Phasen als strikt sequentiell angesehen, wobei in Fehlerfällen in vorhergehende Phasen zurück gewechselt wird. Dieses ist in Abbildung 3.1 durch schwarz eingefärbte Pfeile dargestellt. Die Phasen müssen somit vollständig abgeschlossen werden, bevor mit der nächsten begonnen werden kann. Tritt in einer Phase jedoch, wie oben bereits erwähnt, ein Fehler auf, so erlaubt dieses Vorgehensmodell ein Zurückspringen in die vorgelagerte Phase, um die entsprechenden Korrekturen durchzuführen. In Abbildung 3.1 ist dieses durch weiße Pfeile dargestellt.
Die Möglichkeit, in die vorgelagerte Phase zurückzuspringen, hat bei der Softwareentwicklung der Integrationsgruppe eine entscheidende Rolle gespielt, da viele Probleme erst in der Phase der Realisierung klar geworden sind und daher ein häufiges Verändern des Designs (der Vorphase) nötig geworden ist. Eine Auflistung dieser Probleme erfolgt in Kapitel 8.

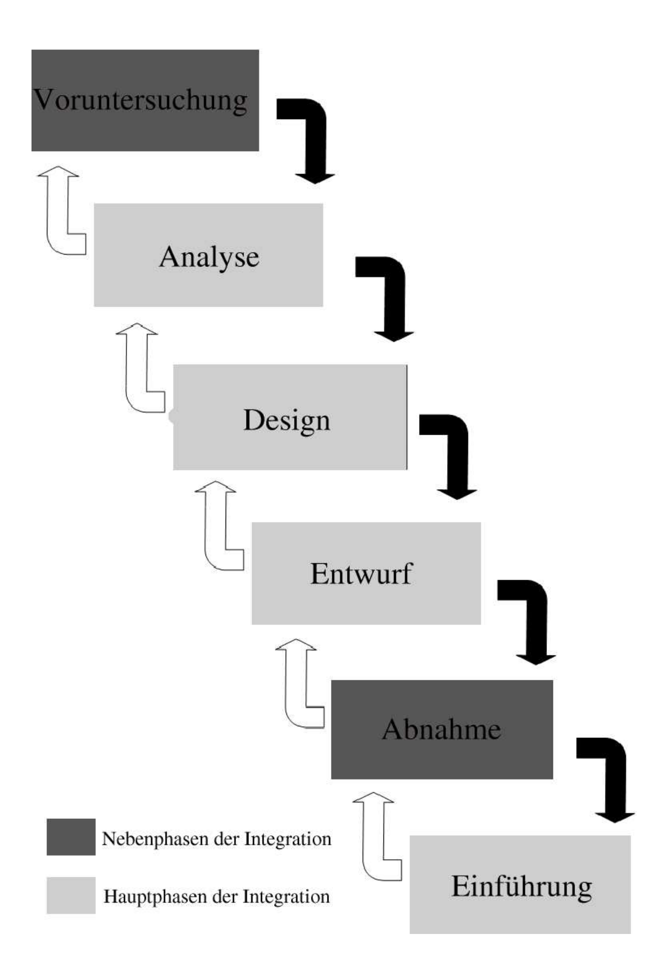


Abbildung 3.1: Wasserfallmodell

3.2 Beschreibung der Phasen der Softwareentwicklung

Die erste Phase des Wasserfallmodelles, die Voruntersuchung, hat bei der Integration nur eingeschränkt stattgefunden, da es nicht nötig war, zu überprüfen, ob der Einsatz der zu entwickelnden Software sinnvoll ist. Die Klärung dieses Problems lag in diesem Szenario bei dem Auftraggeber.
Die erste Aufgabe im Integrationsprojekt kann daher in der Analyse gesehen werden, welches der zweiten Phase im Wasserfallmodell entspricht. In dieser Phase geht es um die Klärung der Fragen, welches

Problem die zu erstellende Software lösen soll und welcher Leistungsumfang vom Auftraggeber gefordert wird. Das Ziel der Analysephase ist es demnach, die Anforderungen an die zu erstellende Software unter Berücksichtigung aller Umgebungsbedingungen exakt zu definieren. Innerhalb des Integrationsprojektes ist dieses in Form der Anforderungsdefinition geschehen, die unter Abschnitt 4.1 einzusehen ist.
Die zweite Aufgabe des Softwareentwurfes entspricht der dritten Phase im Wasserfallmodell, dem Design. Hierbei sind die in der Anforderungsdefinition (siehe Abschnitt 4.1) definierten funktionalen und nicht-funktionalen Anforderungen genauer spezifiziert worden. Der Entwurf (siehe Kapitel 5) stellt das Ergebnis dieser Phase dar.
Nach der Designphase sind die Ergebnisse des Entwurfes in einem Programm umgesetzt worden. Dieses entspricht im Wasserfallmodell der vierten Phase, der Realisierung. Während in der Entwurfsphase noch programmiersprachenunabhängig geplant worden ist, ist hier die konkrete Umsetzung in der Programmiersprache Java erfolgt. Das Resultat dieser Phase bildet der Sourcecode des Programms sowie der darin enthaltene Javadoc-Dokumentationscode.
Die vorletzte Phase des Wasserfallmodelles stellt die Abnahme dar. Diese erfolgt anhand der Kriterien aus der Anforderungsdefinition (siehe Abschnitt 4.1) durch den Auftraggeber. Dieser erhält die oben genannte Dokumentation zur besseren Nachvollziehbarkeit des Vorgehens während der Integration.
Den letzten Abschnitt unseres Projektes stellt die Einführung der fertiggestellten Software dar. Die Einführung entspricht der letzten Phase im Wasserfallmodell. Zur Anwendung kommt die Software dabei in der Analysephase des Gesamtprojektes. Für die leichtere Handhabung seitens der Analysegruppe und für mögliche Erweiterungen ist ein Handbuch (siehe Abschnitt 6.1) erstellt worden.
Im Folgenden werden die, in der Abbildung 3.1 als helle Kästen dargestellten, Hauptphasen (Analyse, Design, Entwurf, Einführung), welche während der Integration im Mittelpunkt gestanden haben, näher erläutert.

4 Objektorientierte Analyse

Die Analyse gliedert sich in drei verschiedene Phasen:

- Erstellung der Anforderungsdefinition
- Erstellung eines statischen Modells (Klassendiagramm)
- Erstellung eines dynamischen Modells (Sequenzdiagramm)

4.1 Anforderungsdefinition

Das Ziel der Analyse ist es, die Wünsche und Anforderungen des Auftraggebers an das zu entwickelnde Softwaresystem zu ermitteln und in dieser Anforderungsdefinition zu formulieren.

4.1.1 Ausgangssituation und Zielsetzung

Ziel der Integration ist es, die Daten aus verschiedenen Händlerdatenbanken in eine Kartenanbieterdatenbank zu importieren. Als Arbeitsgrundlage dienen die bereits angelegten Datenbanken der zwei Händler und die des Kartenanbieters. Die Überlegung, ein zusätzliches Datenbankschema zu erstellen, welches als Zwischendatenbank zwischen den Händler- und Kartenanbieterdatenbanken fungieren soll, wurde aufgrund des zusätzlichen, hohen Arbeitsaufwands nicht implementiert. Ein Vorteil einer solchen Zwischendatenbank wäre die einfachere Erweiterbarkeit des Szenarios bzgl. mehrerer Kartenanbieter. Mit existierender Zwischendatenbank müsste für einen zusätzlichen Kartenanbieter lediglich ein Loader zur Zwischendatenbank implementiert werden und nicht ein Loader pro existierendem Händler. Nachteil einer Zwischendatenbank wäre, dass im Falle nur eines Kartenanbieters ein zusätzlicher Loader von der Zwischendatenbank zum Kartenanbieter geschrieben werden müsste, was eine erhebliche Steigerung des Arbeitsaufwandes darstellen würde. Zudem entstünden erhöhte Kosten durch redundante Datenhaltung und zusätzlichen administrativen Aufwand. Es würde sich die Frage stellen, ob einer der

Händler oder der Kartenanbieter für die Verwaltung der Zwischendatenbank verantwortlich ist.

Systemeinsatz und Systemumgebung

Als Datenbank dient eine Oracle-Datenbank, die unter Solaris läuft. Die Loader werden in Java implementiert, woraus eine Plattformunabhängigkeit resultiert, da das Ausführen auf geeigneten Systemen mit einer Java-Laufzeitumgebung gewährleistet wird.

Funktionale Anforderungen

Es werden alle Daten aus den beiden Händlerdatenbanken ausgelesen, um sie anschließend in die Datenbank des Kartenanbieters einzufügen. Es besteht die Möglichkeit, per Kommandozeilenparameter Einfluss auf die Integration zu nehmen. Dabei können die Namen der Quell- und Zieldatenbank, die erforderlichen Zugangsdaten zu den Datenbanken, der Treiber, der zum Zugriff auf die Datenbank nötig ist sowie der Name des Rechensystems, auf dem die Datenbank läuft, übergeben werden. Eine genauere Schnittstellendefinition erfolgt im Gliederungspunkt Benutzerschnittstellen.
Das Programm besteht aus insgesamt vier Klassen, wobei eine Klasse für die Datenbankverbindung, eine Klasse zur Steuerung des Programmablaufs und jeweils eine Klasse für den Import der Daten eines Händlers zuständig ist. Das Schema für die Datenübertragung zwischen Händlerdatenbanken und Kartenanbieterdatenbank wird in Tabellenform im Anhang dieses Endberichtes dargestellt.

Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen für das System sind

- Portabilität (Plattformunabhängigkeit)
- Stabilität (leichtes Abfangen von Fehlern)
- Benutzerfreundlichkeit (Übergabe von Steuerungsparametern)
- Wart- und Erweiterbarkeit (Übersichtliche Programmstruktur und Dokumentation)
- Sicherheit (Zugriff auf die Datenbanken nur für berechtigte Nutzer)

Großer Wert wird zudem auf eine umfassende Dokumentation des Systems gelegt.

Benutzerschnittstellen

Das Programm lässt sich aus einer Kommandozeile mit Kommandozeilenparametern starten. So können die folgenden acht Parameter int sourceDB, int targetDB, String hostNameSDB, String driverSDB, String userSDB, String passwdSDB, String hostNameTDB, String driverTDB, String userTDB und String passwdTDB optional übergeben werden. In diesem Falle werden die Parameter für die Integration genutzt. Weiterhin existiert die Möglichkeit, nur die sechs Kommandozeilenparameter int sourceDB, int targetDB, String userSDB, String passwdSDB, String userTDB und String passwdTDB zu übergeben. Bei dieser Lösung werden die Werte für hostNameSDB, hostNameTDB sowie driverSDB und driverTDB fest im Programm codiert. Schließlich kann bei initialem Import auch der Kommandozeilenparameter init aufgerufen werden, um die Kartenanbieterdatenbank auf die Importe der Daten aus den Händlerdatenbanken vorzubereiten (Anlegen von Werten für LOV-Tabellen usw.). Auf das Auftreten von Fehlern soll das Programm mit Programmabbruch und Ausgabe eines Fehlertextes reagieren. Folgende Fehler können auftreten:

1. Fehlerhafte Zugangsdaten. Verhalten: kein Verbindungsaufbau.
2. Fehler in der Datenbank (z.B. Attributname geändert). Verhalten: Angabe der Fehlerquelle.
3. Verbindungsabbruch. Verhalten: Fehlermeldung

Dokumentationsanforderungen

Die Dokumentation des Programms umfaßt folgende Punkte:

- Anforderungsdefinition (dieses Dokument)
- Entwurf
- Dokumentierter Sourcecode
- System-Dokumentation bestehend aus Benutzer- und Developer-Handbuch

Die Dokumentation soll in elektronischer Form erfolgen.

Abnahmekriterien

Die funktionalen Anforderungen sollen erfüllt werden; nicht-funktionale Anforderungen sollen möglichst vollständig implementiert werden. Sämtliche für den Kartenanbieter relevanten Daten aus den Händlerdatenbanken sollen von dem Programm in die Kartenanbieterdatenbank übernommen werden. Der Kartenanbieter hat dabei keine Möglichkeit auf die Datenauswahl Einfluss zu nehmen, da diese fest in den Methoden der Importklassen codiert ist. Somit ist auch keine Metadatensteuerung, die prinzipiell möglich wäre, vorgesehen.

4.2 Statische Analyse: Klassendiagramm

Die Klassenbeschreibung umfasst die Beschreibung der Klassen Loader, DBConnection, ImportHaendler1, ImportHaendler2, InitData und CleanDoubles. Die Klasse DBConnection gehört zur Datenbankschicht und ergänzt die JDBC-Klassen, die von Java zur Verfügung gestellt werden. Die übrigen Klassen werden der Anwendungsschicht zugerechnet. Der Aufbau und die Beziehungen der Klassen untereinander sind in Abbildung 4.1 dargestellt.

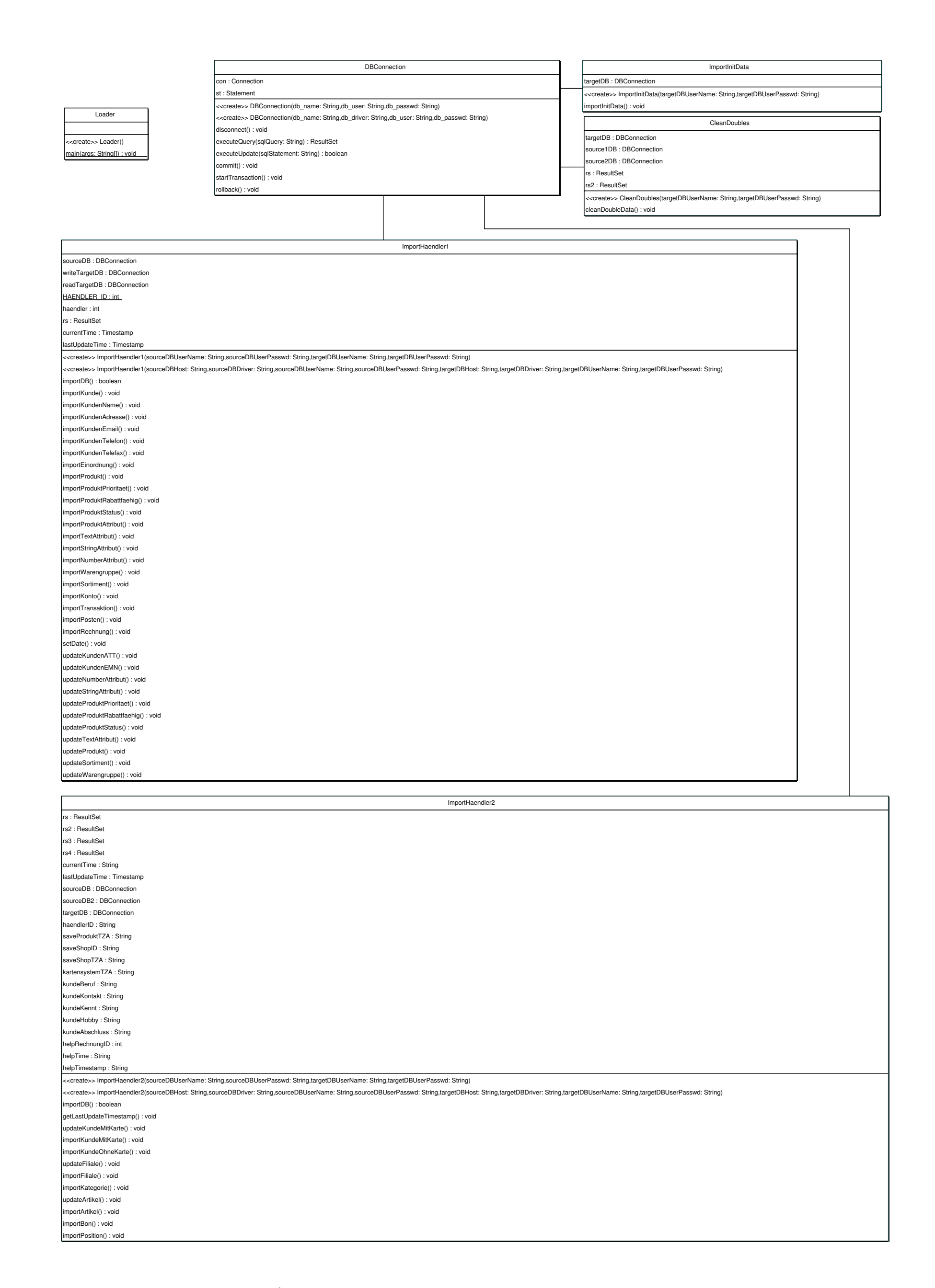


Abbildung 4.1: Klassendiagramm

4.3 Dynamische Analyse: Sequenzdiagramm

Die Dynamik des Systems wird durch das Sequenzdiagramm in Abbildung 4.2 für den wesentlichen Anwendungsfall beschrieben. Es wird sich hierbei auf einen exemplarischen Fall beschränkt, in dem die Klasse ImportHaendler1 aufgerufen wird. Im Falle eines Aufrufs von ImportHaendler2 verlaufen die Klassenaufrufe analog.

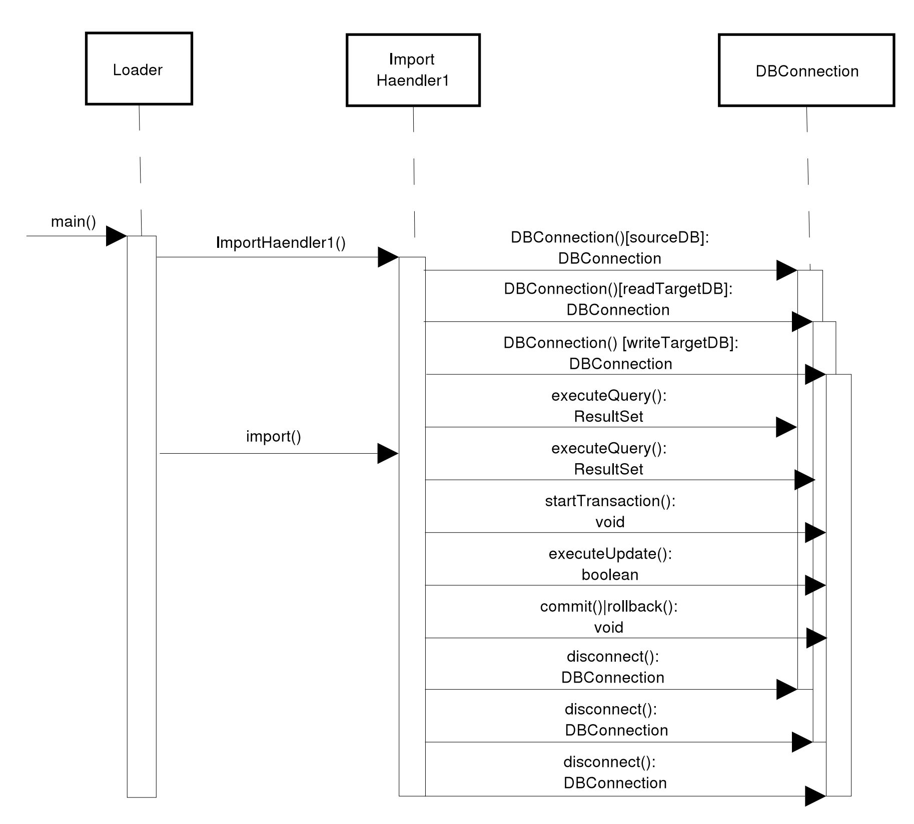


Abbildung 4.2: Sequenzdiagramm

5 Objektorientierter Entwurf

Dieser Entwurf beschreibt den Aufbau eines Softwaresystems zum Datenaustausch zwischen zwei Datenbankschemata, das die in der Anforderungsdefinition gestellten Bedingungen erfüllt. Er enthält die Architektur des Gesamtsystems, die aus einer Zwei-Schichten-Architektur abgeleitet ist.

5.1 Architektur

Die Zwei-Schichten-Architektur des Gesamtsystems ist in eine Anwendungs- und eine Datenbankschicht unterteilt.
Die Datenbankschicht umfasst die bereits implementierten Datenbanken der zwei Händler des ersten Teilprojektes und die des Kartenanbieters, sowie die von der Integrationsgruppe erstellten Klassen, die für eine Verbindung mit der Datenbank zuständig sind. Die Anwendungsschicht umfasst alle weiteren erstellten Klassen.

5.2 Designentscheidungen

Während der Entwicklung des Teilprojekts Integration wurden mehrere Änderungen am Datenbankschema des Kartenanbieters und an den Klassen der Integration vollzogen, die auf grundlegenden Designentscheidungen beruhen. Dazu gehört der Verzicht auf die Implementierung einer Zwischendatenbank (siehe Kapitel 4.1.1) aus Komplexitäts- und Zeitgründen sowie das Einfügen von HaendlerIDs in mehreren Entitäten, um eine Zuordnung der Daten zu einzelnen Händlern zu ermöglichen. Das Einfügen der Attribute *Status*, *Aktualisierungsdatum* und *Fehlerzeitpunkt* in die Entität *Haendler* war notwendig, um die Identifizierung zu importierender Datensätze in den Händlerdatenbanken zu ermöglichen. Das aus den Änderungen resultierende Datenbankschema kann in den Abbildungen 5.1 bis 5.5 eingesehen werden.

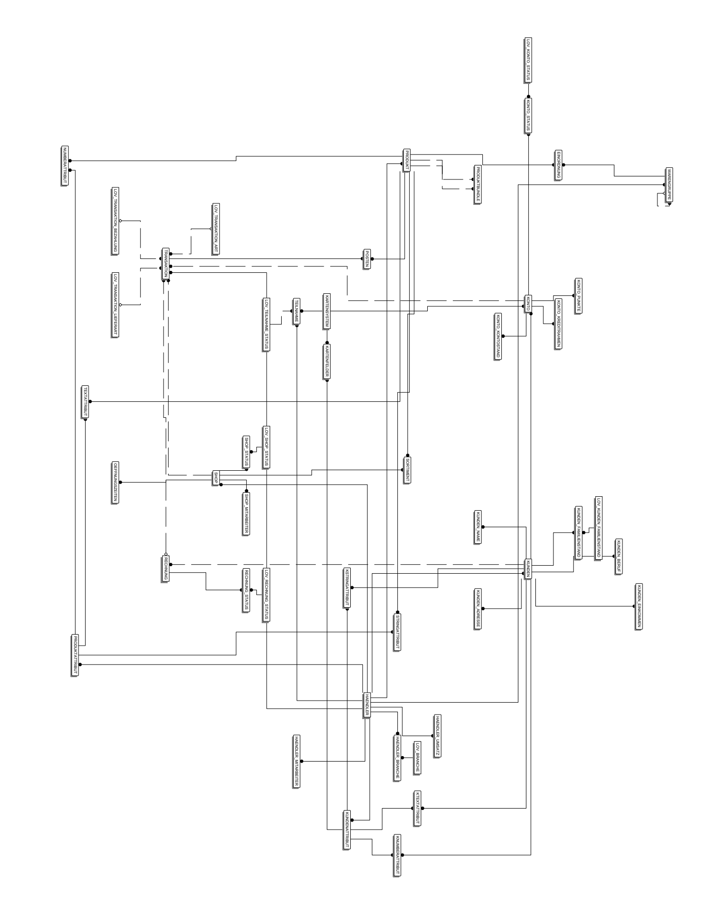


Abbildung 5.1: Angepasstes Datenbankschema des Kartenanbieters

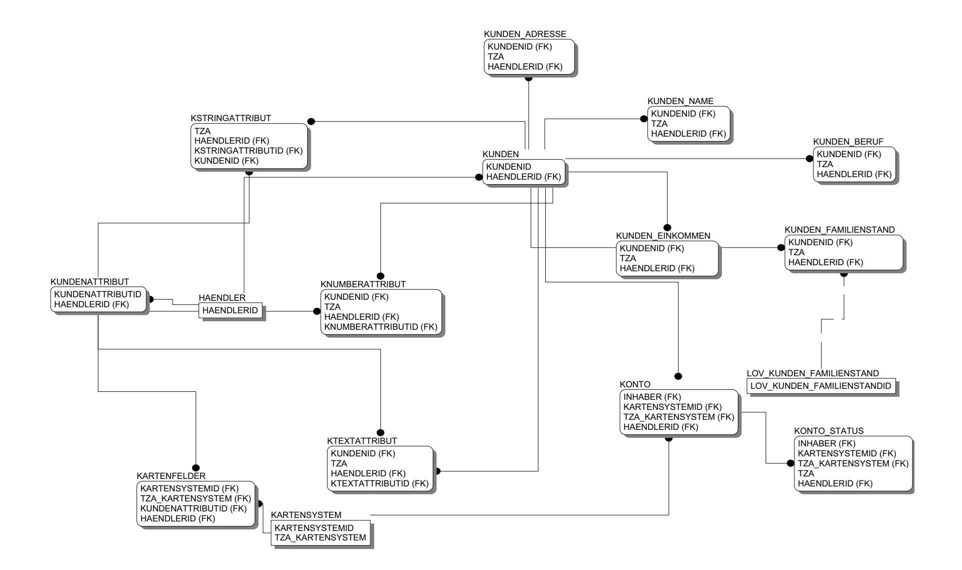


Abbildung 5.2: Subject Area Kunden

Der Verzicht auf eine grafische Benutzungsoberfläche erfolgte aus der Überlegung, dass die Interaktion mit dem Benutzer Aufgabe einer anderen Komponente des gesamten Softwaresystems der Projektgruppe sein wird. Eine weitere GUI würde jedoch eventuell mit dieser Benutzerschnittstelle interferieren. Stattdessen ist das Programm aus einer Shell heraus mit Parameterübergabe aufrufbar. Dafür wurden den beiden Händlern IDs zugeordnet, die dem Programm beim Aufruf übergeben werden, so dass der Import entweder auf Händler 1 oder Händler 2 ausgerichtet ist. Das Programm wird von der Kommandozeile aus folgendermaßen aufgerufen:

$java Loader [parameter]

Dem Kartenanbieter ist ebenfalls eine ID zugeordnet, die nach gleichem Muster funktioniert, auch wenn nur ein Kartenanbieter vorhanden ist. Abschließend wurden, um eine bessere Übersicht und Wartbarkeit der Klassen *ImportHaendler1* und *ImportHaendler2* zu erreichen, die monolithischen *ImportDB*-Methoden in eine Vielzahl kleinerer Methoden aufgespalten. Diese sind jeweils auf den Import der Daten einzelner Entitäten eines Datenbankschemas ausgelegt.
Da es sich bei der Integration ohnehin um ein relativ einfaches System handelt, wurde auf die Verwendung von Interfaces verzichtet.

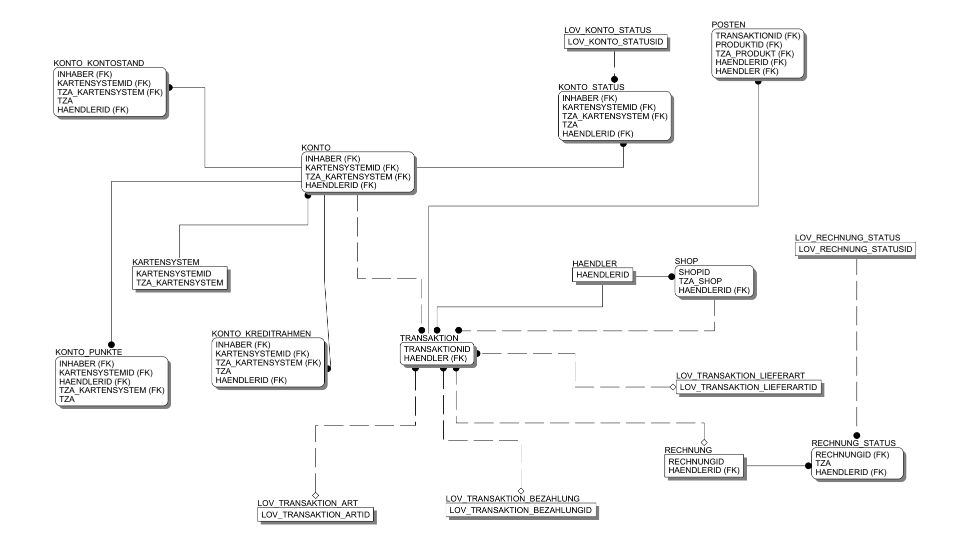


Abbildung 5.3: Subject Area Transaktionen

5.3 Klassenbeschreibung

Die Klassenbeschreibung, dargestellt in Abbildung 5.1, umfasst die Beschreibung der Klassen *Loader*, *DBConnection*, *ImportHaendler1*, *ImportHaendler2*, *InitData* und *CleanDoubles*. Die Klasse *DBConnection* gehört zur Datenbankschicht und ergänzt die JDBC-Klassen, die von Java zur Verfügung gestellt werden. Die übrigen Klassen werden der Anwendungsschicht zugerechnet.

5.3.1 Anwendungsklassen

Die Anwendungsklassen sind im wesentlichen für die Erstellung von SQL-Queries und -Statements entwickelt worden. Außerdem bieten sie Funktionen zur Bereinigung der Datenbank des Händlers 1 von inkonsistenten Daten sowie zur Initialisierung der Kartenanbieterdatenbank.

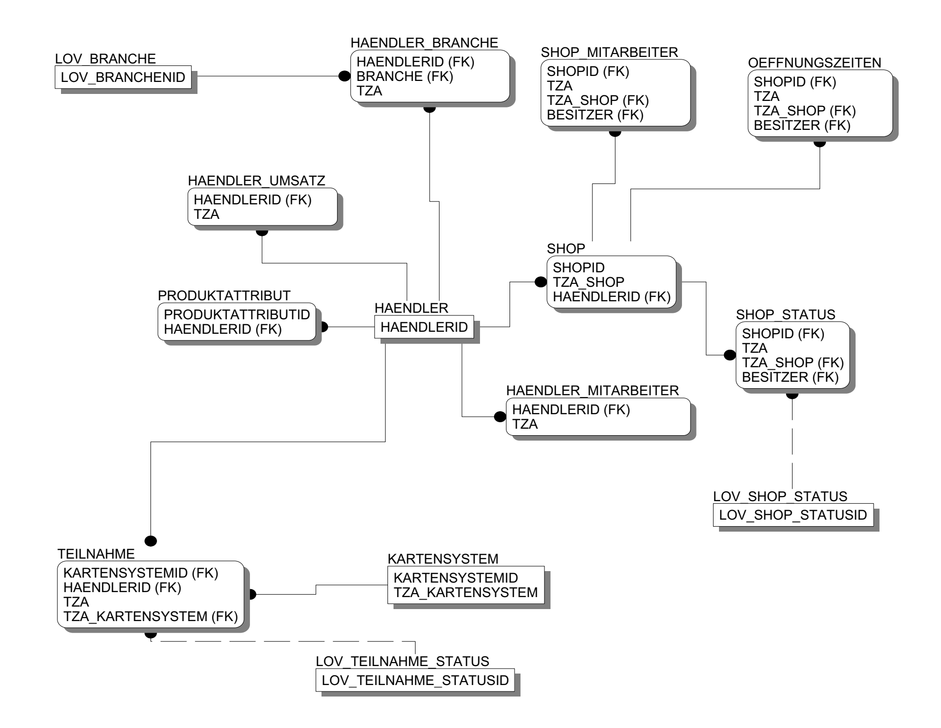


Abbildung 5.4: Subject Area Händler

Klasse Loader

Name
Loader

Kurzbeschreibung
Die Klasse *Loader* ist die Hauptklasse des Programms. Sie wird mit Parametern aufgerufen, die über die Kommandozeile übergeben werden können. Folgende Parameter werden akzeptiert:

- Diese Parameter dienen zum Verbindungsaufbau für einen Import und müssen in der angegebenen Reihenfolge übergeben werden:

  1. *int sourceDB* Die Angabe der Quelldatenbank erfolgt als Integer-Wert. Die Codierung erfolgt nach folgendem Schema:

     – *1* entspricht dem Import der Daten von Händler 1,

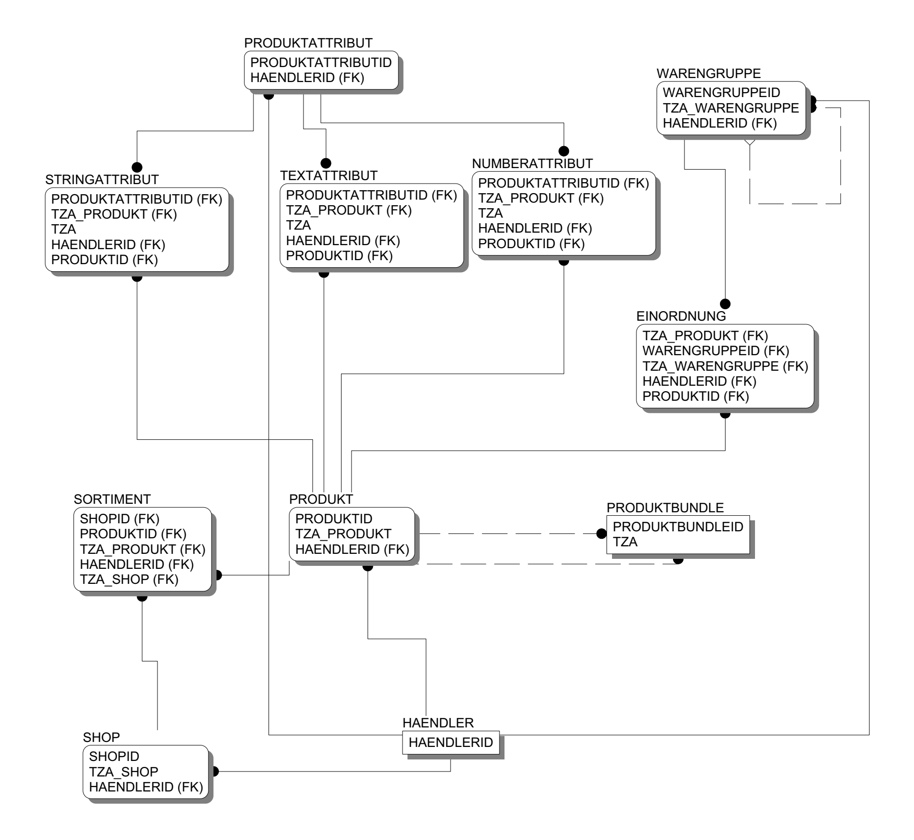


Abbildung 5.5: Subject Area Produkte

der im Szenario der ersten Teilaufgabe implementiert wurde.

- *2* entspricht dem Import der Daten von Händler 2, der im Szenario der ersten Teilaufgabe implementiert wurde.

2. *int targetDB* Die Angabe der Zieldatenbank erfolgt als Integer-Wert. Die Codierung erfolgt nach folgendem Schema:
   - *1* entspricht dem Export der Daten zum Kartenanbieter des Szenarios, welches in der ersten Teilaufgabe implementiert wurde.
3. *String sourceDBHost* gibt die Adresse der Quelldatenbank an.

4. *String sourceDBDriver* gibt den Treiber der Quelldatenbank an.
5. *String userSDB* beinhaltet das User-Login für die Quelldatenbank.
6. *String passwdSDB* hält das Passwort für die Quelldatenbank.
7. *String targetDBHost* gibt die Adresse der Zieldatenbank an.
8. *String targetDBDriver* gibt den Treiber der Zieldatenbank an.
9. *String userTDB* beinhaltet das User-Login für die Zieldatenbank.
10. *String passwdTDB* hält das Passwort für die Zieldatenbank.

Alternativ können die Parameter *sourceDBHost*, *sourceDBDriver*, *targetDBHost* und *targetDBDriver* weggelassen werden. In diesem Fall werden diese Parameter durch fest im *Loader* codierte Werte ersetzt. Diese Werte sind

– für sourceDBHost: „jdbc:oracle:thin:@power2.offis.uni-oldenburg.de:1521:power2“
– für sourceDBDriver: „oracle.jdbc.driver.OracleDriver“
– für targetDBHost: „jdbc:oracle:thin:@power2.offis.uni-oldenburg.de:1521:power2“
– für targetDBDriver: „oracle.jdbc.driver.OracleDriver“

- *-init targetDBHost targetDBDriver userTDB passwdTDB*
  Ein Aufruf mit diesen Parametern gibt an, dass die Klasse *InitData* ausgeführt wird. Diese Klasse dient dazu, initial einige Tabellen in der angegebenen Datenbank mit Daten zu füllen.

  In der Beschreibung der Klasse *ImportInitData* (siehe Kapitel 5.3.1) wird näher auf diese Funktionalität eingegangen.
- *-cleanDoubles sourceDBHost sourceDBDriver userSDB passwdSDB*
  Ein Aufruf mit diesen Parametern gibt an, dass die Klasse *CleanDoubles* ausgeführt wird, um die angegebene Quelldatenbank

von Fehlern zu bereinigen.
In der Beschreibung der Klasse *CleanDoubles* (siehe Kapitel 5.3.1)wird näher auf diese Funktionalität eingegangen.

Diese Klasse ruft je nach Parameterübergabe die entsprechenden Klassen *ImportHaendler1* und *ImportHaendler2* auf.

Oberklassen
keine

Unterklassen
keine

Attribute
keine

Beziehungen
Diese Klasse erzeugt entsprechend der übergebenen Parameter Objekte vom Typ *ImportHaendler1*, *ImportHaendler2*, *CleanDoubles* und *InitData*.

Methoden/Dienste

1. *public static void main(String args[])*
   Die *Main*-Methode ist die Hauptschleife der Klasse.

2. *public Loader()*
   Ein leerer Konstruktor, der lediglich zum Erzeugen von Instanzen der Klasse benötigt wird.

Klasse ImportHaendler1

Name
ImportHaendler1

Kurzbeschreibung
Die Klasse *ImportHaendler1* führt den Import der Daten aus der Händlerdatenbank von Händler 1 in die Datenbank des Kartenanbieters durch. Es werden SQL-Queries und -Statements generiert, mit denen die Daten übertragen werden. Des Weiteren werden Konvertierungen vorgenommen und fehlende Daten mit Standardwerten

ergänzt. Die Übertragung von neuen Datensätzen in die Zieldatenbank geschieht durch Verwendung sogenannter *insert*-Methoden. Bereits in der Zieldatenbank vorhandene Datensätze werden mittels sogenannter *update*-Methoden aktualisiert, falls sich die Datensätze in der Quelldatenbank ändern sollten.

Oberklassen
keine

Unterklassen
keine

Konstanten

1. *HAENDLER_ID* hat den Wert 1.

Attribute

1. *DBConnection sourceDB*
   Datenbankverbindung zur Händlerdatenbank von Händler1.

2. *DBConnection readTargetDB*
   Datenbankverbindung für lesende Zugriffe auf die Datenbank vom Kartenanbieter.

3. *DBConnection writeTargetDB*
   Datenbankverbindung für schreibende Zugriffe auf die Datenbank vom Kartenanbieter.

4. *Timestamp currentTime*
   In diesem Attribut wird der Zeitpunkt des Programmstarts gespeichert.

5. *Timestamp lastUpdateTime*
   In diesem Attribut wird der Zeitpunkt des letzten Import gespeichert, welcher aus der Tabelle *Cardprovider.Haendler* ausgelesen wird.

6. *ResultSet rs*
   In diesem Attribut werden die Ergebnisse von Datenbankabfragen gespeichert.

Beziehungen
Die Klasse *ImportHaendler1* wird von der Klasse *Loader* aufgerufen, um die Daten zu importieren. Sie verwendet die Klasse *DBConnection*, um eine Datenbankverbindung aufzubauen.

Methoden/ Dienste

1. *ImportHaendler1(String sourceDBHost, String sourceDBDriver, String sourceDBUserName, String sourceDBUserPasswd, String targetDBHost, String targetDBDriver, String targetDBUserName, String targetDBUserPasswd)*
   Konstruktor, der ein Objekt vom Typ *ImportHaendler1* instantiiert.

2. *boolean importDB()*
   importiert die Daten der Datenbank von Händler 1 in die Datenbank des Kartenanbieters unter Berücksichtigung der Unterschiede der beiden Schemata. Ruft die Methoden für den Import und die Aktualisierung der Daten aus den einzelnen Tabellen auf (mit anderen Worten: alle anderen Methoden). Setzt je nach Fortschritt des Imports den Wert für das Attribut *Status* in der Tabelle *Haendler* des Kartenanbieters. Liefert *true* zurück, falls der Import ohne Probleme abschließt, ansonsten *false*.

3. *void importEinordnung()*
   importiert die Daten in die Tabelle *Einordnung* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

4. *void importKonto()*
   importiert die Daten in die Tabelle *Konto* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

5. *void importKunden()*
   importiert die Daten in die Tabelle *Kunden* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

6. *void importKundenAdresse()*
   importiert die Daten in die Tabelle *Kunden_Adresse* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

7. *void importKundenEMail()*
   importiert die Daten in die Tabelle *KStringAttribut* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

8. *void importKundenName()*
   importiert die Daten in die Tabelle *Kunden_Name* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

9. *void importKundenTelefax()*
   importiert die Daten in die Tabelle *KStringAttribut* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

10. *void importKundenTelefon()*
    importiert die Daten in die Tabelle *KStringAttribut* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

11. *void importNumberAttribut()*
    importiert die Daten in die Tabelle *NumberAttribut* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

12. *void importPosten()*
    importiert die Daten in die Tabelle *Posten* des Kartenanbieterschemas. Wird innerhalb von der Methode importDB aufgerufen.

13. *void importProdukt()*
    importiert die Daten in die Tabelle *Produkt* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

14. *void importProduktAttribut()*
    importiert die Daten in die Tabelle *ProduktAttribut* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

15. *void importProduktPrioritaet()*
    importiert die Daten in die Tabelle *StringAttribut* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

16. *void importProduktRabattfaehig()*
    importiert die Daten in die Tabelle *StringAttribut* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

17. *void importProduktStatus()*
    importiert die Daten in die Tabelle *StringAttribut* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

18. *void importRechnung()*
    importiert die Daten in die Tabelle *Rechnung* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

19. *void importSortiment()*
    importiert die Daten in die Tabelle *Sortiment* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

20. *void importStringAttribut()*
    importiert die Daten in die Tabelle *StringAttribut* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

21. *void importTextAttribut()*
    importiert die Daten in die Tabelle *TextAttribut* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

22. *void importTransaktion()*
    importiert die Daten in die Tabelle *Transaktion* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

23. *void importWarengruppe()*
    importiert die Daten in die Tabelle *Warengruppe* des Kartenanbieterschemas. Wird von der Methode *importDB* aus aufgerufen.

24. *void setDate()*
    Die Methode setzt nach einem erfolgreichen Import das Attribut *Aktualisierungsdatum* in *Cardprovider.Haendler* auf den Wert von *currentTime*.

25. *void updateKundenATT()*
    Die Methode aktualisiert die tze-Werte von Datensätzen in den Tabellen *Kunden_Adresse* und *KStringAttribut* (bezüglich *Telefon* und *Telefax*) der Kartenanbieterdatenbank, falls diese seit dem letzten Import in der Quelldatenbank gesetzt wurden.

26. *void updateKundenEMN()*
    Die Methode aktualisiert die tze-Werte von Datensätzen in den Tabellen *Kunden_Name* und *KStringAttribut* (bezüglich *Email*) der Kartenanbieterdatenbank, falls diese seit dem letzten Import in der Quelldatenbank gesetzt wurden.

27. *void updateNumberAttribut()*
    Die Methode aktualisiert die tze-Werte von Datensätzen in *Cardprovider.NumberAttribut*, falls diese seit dem letzten Import in der Quelldatenbank gesetzt wurden.

28. *void updateTextAttribut()*
    Die Methode aktualisiert die tze-Werte von Datensätzen in der Kartenanbieterdatenbank, falls diese seit dem letzten Import in der Quelldatenbank gesetzt wurden.

29. *void updateStringAttribut()*
    Die Methode aktualisiert die tze-Werte von Datensätzen in der Kartenanbieterdatenbank, falls diese seit dem letzten Import in der Quelldatenbank gesetzt wurden.

30. *void updateProdukt()*
    Die Methode aktualisiert die tze-Werte von Datensätzen in der Kartenanbieterdatenbank, falls diese seit dem letzten Import in der Quelldatenbank gesetzt wurden.

31. *void updateProduktPrioritaet()*
    Die Methode aktualisiert die tze-Werte von Datensätzen in der Kartenanbieterdatenbank, falls diese seit dem letzten Import in der Quelldatenbank gesetzt wurden.

32. *void updateProduktRabattfaehig()*
    Die Methode aktualisiert die tze-Werte von Datensätzen in der Kartenanbieterdatenbank, falls diese seit dem letzten Import in der Quelldatenbank gesetzt wurden.

33. *void updateProduktStatus()*
    Die Methode aktualisiert die tze-Werte von Datensätzen in der Kartenanbieterdatenbank, falls diese seit dem letzten Import in der Quelldatenbank gesetzt wurden.

34. *void updateSortiment()*
    Die Methode aktualisiert die tze-Werte von Datensätzen in der Kartenanbieterdatenbank, falls diese seit dem letzten Import in der Quelldatenbank gesetzt wurden.

35. *void updateWarengruppe()*
    Die Methode aktualisiert die tze-Werte von Datensätze in der Kartenanbieterdatenbank, falls diese seit dem letzten Import in der Quelldatenbank gesetzt wurden.

Klasse ImportHaendler2

Name
ImportHaendler2

Kurzbeschreibung
Die Klasse *ImportHaendler2* führt den Import der Daten aus der Händlerdatenbank des Händlers 2 in die Datenbank des Kartenanbieters durch. Es werden SQL-Queries und -Statements generiert, mit denen die Daten von der Händlerdatenbank in die Kartenanbieterdatenbank übertragen werden. Des Weiteren werden Konvertierungen vorgenommen und fehlende Daten mit Standardwerten ergänzt. Die Übertragung von neuen Datensätzen in die Zieldatenbank geschieht durch Verwendung sogenannter *insert*-Methoden. Bereits in der Zieldatenbank vorhandene Datensätze werden mittels sogenannter *update*-Methoden aktualisiert, falls sich die Datensätze in der Quelldatenbank ändern sollten.

Oberklassen
keine

Unterklassen
keine

Attribute

1. *DBConnection sourceDB*
   Datenbankverbindung zur Datenbank von Händler 2.

2. *DBConnection sourceDB2*
   Datenbankverbindung zur Datenbank des Kartenanbieters nur für lesende Zugriffe.

3. *DBConnection targetDB*
   Datenbankverbindung zur Datenbank des Kartenanbieters nur für schreibende Zugriffe.

4. *ResultSet rs*
   Dieses Attribut enthält das aktuelle Ergebnis einer Datenbankabfrage.

5. *ResultSet rs2*
   Ein weiteres Attribut, welches das aktuelle Ergebnis einer Datenbankabfrage enthalten kann.

6. *ResultSet rs3*
   Noch ein Attribut, welches das aktuelle Ergebnis einer Datenbankabfrage enthalten kann. Importiert die Daten in die Tabelle Warengruppe der Kartenanbieterdatenbank.

7. *ResultSet rs4*
   Auch dieses Attribut kann aktuelle Ergebnisse von Datenbankabfragen enthalten. Durch die Verwendung von mehreren ResultSets kann auf die Ergebnisse mehrere Abfragen zugegriffen werden.

8. *String currentTime*
   In diesem Attribut wird der Zeitpunkt des Programmstarts gespeichert.

9. *Timestamp lastUpdateTime*
   In diesem Attribut wird der Wert des Attributs Aktualisierungsdatum aus der Tabelle *Haendler* der Kartenanbieterdatenbank gespeichert.

10. *String haendlerID*
    Speichert die ID des Händlers; hier wird das Attribut konstant auf 2 gesetzt.

11. *String helpRechnungID*
    Hilfsvariable für die *RechnungsID*.

12. *String helpTime*
    Hilfsvariable zur Speicherung des Grunddatums
    (1.1.1900 1:00:00.0)

13. *String helpTimestamp*
    Hilfsvariable, die benutzt wird, wenn in den Quelldaten ein Datum nicht vorhanden ist, welches aber für den Import notwendig ist. Basiert auf *helpTime*.

14. *String kartensystemTZA*
    Hilfsvariable zur Speicherung eines Datums, welches für das Anlegen eines Kartensystems verwendet wird.

15. *String kundeBeruf*
    Hilfsvariable zur Zwischenspeicherung von ausgelesenen Werten. In der Variable werden Berufsbezeichnungen gespeichert, die aus der LOV-Tabelle *Beruf* des Datenbankschemas von Händler 2 stammen.

16. *String kundeKontakt*
    Hilfsvariable zur Zwischenspeicherung von ausgelesenen Werten. In der Variable werden die Kontaktbeschreibungen gespeichert, die aus der LOV-Tabelle *Weiteren_Kontakt* des Datenbankschemas von Händler 2 stammen.

17. *String kundeKennt*
    Hilfsvariable zur Zwischenspeicherung von ausgelesenen Werten. In der Variable werden die Beschreibungen gespeichert, die aus der LOV-Tabelle *Kennt_Ueber* des Datenbankschemas von Händler 2 stammen.

18. *String kundeHobby*
    Hilfsvariable zur Zwischenspeicherung von ausgelesenen Werten. In der Variable werden die Hobbybeschreibungen gespeichert, die aus der LOV-Tabelle *Hobbys* des Datenbankschemas von Händler 2 stammen.

19. *String kundeAbschluss*
    Hilfsvariable zur Zwischenspeicherung von ausgelesenen Werten. In der Variable werden die Beschreibungen gespeichert, die

aus der Tabelle *Abschluss* des Datenbankschemas von Händler 2 stammen.

20. *String saveProduktTZA*
Hilfsvariable zur Zwischenspeicherung von ausgelesenen Werten, in diesem Fall den bestimmten ProduktIDs zugeordneten tza-Werten.

21. *String saveShopID*
Hilfsvariable zur Zwischenspeicherung von ausgelesenen Werten, in diesem Fall den IDs der Filialen des Händlers.

22. *String saveShopTZA*
Hilfsvariable zur Zwischenspeicherung von ausgelesenen Werten, in diesem Fall den einzelnen Shops zugeordneten tza-Werten.

Beziehungen
Die Klasse *ImportHaendler2* wird von der Klasse *Loader* aufgerufen, um die Daten zu importieren. Sie verwendet die Klasse *DBConnection*, um eine Datenbankverbindung aufzubauen.

Methoden/ Dienste

1. *ImportHaendler2(String sourceDBHost, String sourceDBDriver, String sourceDBUserName, String sourceDBUserPasswd, String targetDBHost, String targetDBDriver, String targetDBUserName, String targetDBUserPasswd)*
Konstruktor, der ein Objekt vom Typ ImportHaendler2 instantiiert.

2. *ImportHaendler2(String sourceDBUserName, String sourceDBUserPasswd, String targetDBUserName, String targetDBUserPasswd)*
Alternativer Konstruktor, der ein Objekt vom Typ ImportHaendler2 instantiiert.

3. *boolean ImportDB()*
importiert die Daten der Datenbank im Datenbankschema von Händler 1 in die Datenbank des Kartenanbieters in Abhängigkeit der übergebenen Parameter. Ruft die Methoden für den Import und die Aktualisierung der Daten aus den einzelnen Tabellen auf (mit anderen Worten: alle anderen Methoden). Setzt

je nach Fortschritt des Imports den Wert für das Attribut *Status* in der Tabelle *Haendler* des Kartenanbieters. Liefert *true* zurück, falls der Import ohne Probleme abschließt, ansonsten *false*.

4. *void getlastUpdateTimestamp()*
   Liest den Wert von *Haendler.Aktualisierungsdatum* aus der Kartenanbieterdatenbank aus.

5. *void importArtikel()*
   Die Methode liest die relevanten Daten aus der Tabelle *Artikel* im Datenbankschema von Händler 2 aus und fügt sie zusammen mit den notwendigen Daten aus verwandten Tabellen in die Tabellen *Produkt*, *Einordnung* und *Sortiment* ein.

6. *void importBon()*
   Die Methode liest die relevanten Daten aus der Tabelle *Bon* im Datenbankschema von Händler 2 aus und fügt sie zusammen mit den notwendigen Daten aus verwandten Tabellen in die Tabellen *Transaktion* und *Rechnung* ein.

7. *void importFiliale()*
   Die Methode liest die relevanten Daten aus der Tabelle *Artikel* im Datenbankschema von Händler 2 aus und fügt sie zusammen mit den notwendigen Daten aus verwandten Tabellen in die Tabelle *Shop* ein.

8. *void importKategorie()*
   Die Methode liest die relevanten Daten aus der Tabelle *Artikel* im Datenbankschema von Händler 2 aus und fügt sie zusammen mit den notwendigen Daten aus verwandten Tabellen in die Tabelle *Warengruppe* ein.

9. *void importKundeMitKarte()*
   Die Methode liest die relevanten Daten aus der Tabelle *Artikel* im Datenbankschema von Händler 2 aus und fügt sie zusammen mit den notwendigen Daten aus verwandten Tabellen in die Tabellen *Kunden*, *Kunden_Namen*, *Kunden_Adresse*, *Kunden_Familienstand*, *Kunden_Beruf*, *Konto*, *KNumberAttribut* und *KStringAttribut* ein.

10. *void importKundeOhneKarte()*
    Die Methode liest die relevanten Daten aus der Tabelle *Artikel*

im Datenbankschema von Händler 2 aus und fügt sie zusammen mit den notwendigen Daten aus verwandten Tabellen in die Tabellen *Kunden* und *Kunden_Adresse* ein.

11. *void importPosition()*
Die Methode liest die relevanten Daten aus der Tabelle *Artikel* im Datenbankschema von Händler 2 aus und fügt sie zusammen mit den notwendigen Daten aus verwandten Tabellen in die Tabelle *Posten* ein.

12. *void updateArtikel()*
Die Methode aktualisiert die tze-Werte der Datensätze in den Tabellen *Produkt* und *Sortiment* im Kartenanbieterschema falls diese seit dem letzten Import gesetzt worden sind.

13. *void updateFiliale()*
Die Methode aktualisiert die tze-Werte der Datensätze in die Tabelle *Shop* des Kartenanbieterschemas, falls diese seit dem letzten Import gesetzt worden sind.

14. *void updateKundeMitKarte()*
Die Methode aktualisiert die tze-Werte der Datensätze in den Tabellen *Kunden_Name*, *Kunden_Adresse*, *Kunden_Familienstand*, *Kunden_Beruf*, *KNumberAttribut* und *KStringAttribut* des Kartenanbieterschemas, falls diese seit dem letzten Import gesetzt worden sind.

Klasse CleanDoubles

Name
CleanDoubles

Kurzbeschreibung
Die Klasse *CleanDoubles* entfernt aus den Quelldaten von Händler 2 Datensätze, deren Attribute in der Quelldatenbank dieselben Werte haben. Diese Datensätze können in die Kartenanbieterdatenbank nicht problemlos übernommen werden, da die Attribute dort als Primärschlüssel verwendet werden. Um Fehler während des Imports zu vermeiden, müssen die inkonsistenten Datensätze vorher aus der Quelldatenbank entfernt werden. Diese Klasse wurde nur zur Bereinigung der Testdaten von Händler 2 entwickelt und ist nur der

Vollständigkeit der von der Klasse *Loader* verwendeten Objekte wegen in diesem Dokument aufgeführt.
Durch das Entfernen der doppelten Datensätze entsteht ein Informationsverlust. Eine Zusammenfassung der doppelten Daten hätte eine sauberere Implementation dargestellt. Darauf wurde jedoch aus Zeitgründen verzichtet.

Oberklassen
keine

Unterklassen
keine

Attribute

1. *DBConnection targetDB*
   Stellt eine Verbindung zur Datenbank dar, die für schreibende Zugriffe genutzt wird.

2. *DBConnection source1DB*
   Stellt eine Verbindung zur Datenbank dar, die für lesende Zugriffe genutzt wird.

3. *DBConnection source2DB*
   Stellt eine weitere Verbindung zur Datenbank dar, die für lesende Zugriffe genutzt wird.

4. *ResultSet rs*
   Enthält das Ergebnis einer aktuellen Datenbankabfrage.

5. *ResultSet rs2*
   Enthält das Ergebnis einer aktuellen Datenbankabfrage.

Beziehungen
Wird von der Klasse *ImportHaendler1* aufgerufen und erzeugt Instanzen vom Typ *DBConnection.*

Methoden/ Dienste

1. *CleanDoubles(String targetDBHost, String targetDBDriver, String targetDBUserName, String targetDBPasswd)*
   Konstruktor der Klasse, erzeugt ein Objekt vom Typ Clean-Doubles.

2. *void cleanDoubleData()*
   Diese Methode führt die Löschvorgänge durch.

Klasse ImportInitData

Name
ImportInitData

Kurzbeschreibung
Die Klasse fügt die Datensätze in die Zieldatenbank ein, die für den weiteren Import benötigt werden, aber nicht aus den Daten der Quelldatenbank erzeugt werden können. Mit dieser Klasse werden die Tabellen *Haendler*, *Kartensystem*, *Kundenattribut*, *LOV_Kunden_Familienstand*, *Produktattribut* und *Shop* im Kartenanbieterdatenbankschema mit Daten gefüllt. Das Vorhandensein dieser Daten ist notwendig, damit neue Datensätze eingefügt werden können.

Oberklassen
keine

Unterklassen
keine

Attribute

1. *DBConnection targetDB*
   Stellt eine Verbindung zur Zieldatenbank dar.

Beziehungen
Wird vom *Loader* instantiiert, wenn der Parameter *-init* angegeben wurde. *ImportInitData* verwendet die Klasse *DBConnection*, um eine Datenbankverbindung aufzubauen.

Methoden/ Dienste

1. *ImportInitData(String targetDBHost, String targetDBDriver, String targetDBUserName, String targetDBPasswd)*
   Konstruktor der Klasse, erzeugt Objekt vom Typ *ImportInitData*.
2. *void ImportInitData()*
   Diese Methode führt die notwendigen Einfügeoperationen auf der Kartenanbieterdatenbank aus.

5.3.2 Datenbankanbindung

Die Datenbankanbindung besteht aus der Klasse *DBConnection*, außerdem werden die Datenbanken von Händler 1, Händler 2 und des Kartenanbieters zu dieser Ebende des Softwareentwurfs hinzugezählt. Die Datenanbindung ist für die Datenhaltung und den Datentranfer von und zu Datenbanken zuständig.

Klasse DBConnection

Name
DBConnection

Kurzbeschreibung
Die Klasse *DBConnection* kapselt eine JDBC-Verbindung zur Oracle-Datenbank. Die Klasse soll eine Verbindung auf- und abbauen können sowie SQL-Statements (Abfragen und Manipulationen) auf der Datenbank ausführen. Die Klasse *Loader* erzeugt Instanzen dieser Klasse, um Verbindungen mit der Quell- und Zieldatenbank aufzubauen.

Oberklassen
keine

Unterklassen
keine

Attribute

1. *Connection con*
   Dieses Attribut stellt die eigentliche Verbindung zur Datenbank dar. Über *con* wird mittels SQL-Queries und -Statements auf die Datenbank zugegriffen.

Beziehungen
DBConnection wird von den Klassen *ImportInitData*, *CleanDoubles*, *ImportHaendler1* und *ImportHaendler2* aufgerufen.

Methoden/Dienste

1. *DBConnection(String db_name, String db_driver, String db_user, String db_passwd)*
   Konstruktor; stellt eine Verbindung zur Datenbank her.

2. *void disconnect()*
   beendet eine Verbindung zur Datenbank.

3. *ResultSet executeQuery(String sqlQuery)*
   führt die in dem übergebenen String enthaltene Anfrage auf der Datenbank aus.

4. *void commit()*
   führt einen *commit* auf die Datenbank aus.

5. *void rollback()*
   verwirft die Änderungen seit dem letzten *commit*.

6. *boolean executeUpdate(String sqlStatement)*
   führt die in dem übergebenen String angegebene Manipulation aus.

7. *void startTransaction()*
   Diese Methode schaltet den Modus *autocommit* aus.

5.4 Dynamikbeschreibung

Die Dynamik des Systems wurde in Kapitel 4.3 behandelt und im wesentlichen durch das Sequenzdiagramm in Abbildung 4.2 beschrieben.

6 Technisches Handbuch

Im Folgenden wird die Nutzung der Klassen des Integrationsschemas anhand eines technischen Handbuches näher erläutert. Es werden Beispiele für die anzugebenden Parameter vorgestellt und es wird erklärt, was die Parameter im Einzelnen bedeuten.

6.1 Benutzerhandbuch

Das Programm *Loader* ist ein kommandozeilengesteuertes Tool für den Transfer von Daten aus Händlerdatenbanken in die Datenbank eines oder mehrerer Kartenanbieter. Das Programm läuft in einer Java-Laufzeitumgebung und wird mit einem Standard-Java-Interpreter und dem Aufruf *java diko.integration.Loader [params]* gestartet. Voraussetzung für den Programmstart ist ein korrekt gesetzter Klassenpfad. Es kann pro Programmaufruf eine Parameterfolge übergeben werden. Folgende Werte können als Parameter übergeben werden:

- *-h*
  Dieser Parameter ruft eine Hilfe zur Benutzung des Programms auf.

- *-init Target_Hostname Target_Driver Target_UserName Target_Passwort*
  Diese Parameterfolge führt die initialen Einfügeoperationen auf der Kartenanbieterdatenbank durch. Es ist u.a. nötig, die Werte einer *lov-Tabelle* (engl. „list of value“) in das Kartenanbieterschema einzufügen. Weiterhin müssen zwei Händlerdatensätze angelegt werden, denen dann die zu importierenden Daten zuzuordnen sind. Außerdem müssen sämtliche dynamischen Attribute angelegt werden, die für den Import der Daten aus den Schemata der Händler notwendig sind. Schließlich benötigt jeder Händler eine Zuordnung zu einem Kartensystem, welches ebenfalls für jeden Händler angelegt werden muss.

- *-cleanDoubles Source_Hostname Source_Driver Source_UserName Source_Passwort*
  Diese Parameterfolge startet eine Methode zur Beseitigung fehlerhafter Daten aus den Quelldaten von Händler 2.

- *HändlerID KartenanbieterID Source_Hostname Source_Driver Source_UserName Source_Password Target_Hostname Target_Driver Target_UserName Target_Password*
  Diese Parameterfolge gibt an, zwischen welchen Datenbanken eine Datenübertragung vorgenommen werden soll. Es sind alle nötigen Verbindungsparameter enthalten.

- *HändlerID KartenanbieterID Source_UserName Source_Password Target_UserName Target_Password*
  Diese Parameterfolge gibt an, zwischen welchen Datenbanken eine Datenübertragung vorgenommen werden soll. Es werden Benutzernamen und Passwörter für Quell- und Zieldatenbanken übergeben. Hostname und Driver sind für Quell- und Zieldatenbank fest codiert, womit dieser spezielle Programmaufruf auch lediglich für diese Datenbanken funktioniert.

Die einzelnen Verbindungsparameter werden bei den Aufrufen wie folgt verwendet:

- *Source_Hostname*
  Hierbei handelt es sich um die genaue Adresse der Quelldatenbank.
  Beispiel:
  „jdbc:oracle:thin:power2.offis.uni-oldenburg.de:1521:power2“

- *Source_Driver*
  Der für die Verbindung zur Quelldatenbank verwendete Treiber.
  Beispiel: „oracle.jdbc.Driver.OracleDriver“

- *Source_UserName*
  Das Login für die Quelldatenbank.
  Beispiel: „haendler1“

- *Source_Password*
  Das Passwort für den Zugriff auf die Quelldatenbank.
  Beispiel: „Passwort“

- *Target_Hostname*
  Hierbei handelt es sich um die genaue Adresse der Zieldatenbank.
  Beispiel:
  „jdbc:oracle:thin:power2.offis.uni-oldenburg.de:1521:power2"

- *Target_Driver*
  Der für die Verbindung zur Zieldatenbank verwendete Treiber.
  Beispiel: „oracle.jdbc.Driver.OracleDriver"

- *Target_UserName*
  Das Login für die Zieldatenbank.
  Beispiel: „cardprovider"

- *Target_Password*
  Das Passwort für den Zugriff auf die Zieldatenbank.
  Beispiel: „MeinGeburtstag"

Vor dem Programmaufruf für den Datentransfer muss zunächst ein einmaliger Programmaufruf von *Loader* mit dem Parameter *-init* erfolgen, da dadurch initial mehrere Tabellen in der Zieldatenbank mit Werten gefüllt werden, die für den Import notwendig sind. Eine genaue Beschreibung der initial zu importierenden Daten findet sich oben in diesem Abschnitt.

6.2 Meilensteine

Zunächst sind folgende Meilensteine festgelegt worden:

1. Fertigstellung der Anforderungsdefinition Integration bis zum 03.03.2003. Dieser Meilenstein ist eingehalten worden.

2. Fertigstellung der Implementation des gesamten Integrationsszenarios bis zum 04.05.2003. Dieser Meilenstein ist bzgl. der Fertigstellung der Software eingehalten worden. Die endgültige Fertigstellung, zu der auch die Erstellung eines Handbuches und der vollständigen Dokumentation zählen, ist bis Ende Mai 2003 erreicht worden.

7 Erweiterbarkeit

In diesem Abschnitt wird darauf eingegangen, wie Erweiterungen des bisherigen Integrationsumfeldes behandelt werden können. Dabei wird unterschieden, ob sich die schon vorhandenen Akteure (Händler 1, Händler 2 und Kartenanbieter) verändern oder ob neue Ziele oder Quellen hinzugefügt werden.

Bei der Erweiterung des bisherigen Integrationsszenarios wird jeweils auf die Akteure einzeln eingegangen.

7.1 Erweiterung der Quelldatenbanken

Werden die Schemata der bisherigen Händlerdatenbanken geändert, so muss überprüft werden, inwieweit sich diese Änderungen bzw. Erweiterungen auf die Integration auswirken. Es muss entschieden werden, ob neue Daten integriert werden können und gegebenenfalls auch importiert werden sollen, d.h. ob diese neuen Daten für eine Analyse von Relevanz sind. Das Kartenanbieterschema wird dabei als gegeben angenommen, was bedeutet, dass dieses nicht, entsprechend den neuen Entitäten oder Attributen in den Quellen, angepasst wird.

Werden weitere Daten in Tabellen eingefügt, in denen bisher schon Daten eingefügt worden sind, so ist die entsprechende *import*-Methode der Tabelle in *importHaendler1* bzw. *importHaendler2* anzupassen. Das gleiche gilt im umgekehrten Fall, wenn Daten, die bisher integriert wurden, nicht mehr in der Quelle zur Verfügung stehen.

7.1.1 Erweiterungen der Schemata der Händler

Die Probleme, die in Zusammenhang mit einer Erweiterung der Quelldatenbank *Händler1* stehen, lassen sich in die folgenden Bereiche untergliedern:

Einfügereihenfolge

Werden neue Tabellen vom Kartenanbieter mit Daten gefüllt, die bisher noch nicht berücksichtigt worden sind, so muss garantiert sein, dass alle notwendigen Daten in der Datenbank bereits eingefügt und durch ein commit bestätigt worden sind. Diese notwendigen Daten lassen sich anhand der Fremdschlüsselbeziehungen herausfinden.

Dynamische Attribute

Die meisten Attribute eines Kunden werden in der Zieldatenbank anders gespeichert als in der Quelldatenbank. Werden neue Kundenattribute gespeichert, so muss eine neue Instanz des Typs *KundenAttribut* manuell angelegt werden und der entsprechende Attributwert als *KString-, KNumber- oder KTextAttribut* abgespeichert werden.
Bei den Produkten verwendet der Händler 1 für viele Attribute die gleiche Speicherstruktur wie der Kartenanbieter. Werden jedoch neue Attribute direkt in der Tabelle *Produkt* angelegt, so müssen diese in gleicher Weise wie die Kundenattribute behandelt werden. Es ist jedoch zu beachten, dass die *ID* einen negativen Wert haben muss, da die positiven *ID*-Werte für die schon in der Quelldatenbank vorhandenen Produktattribute reserviert sind.
Bei den Produkten des Händler 2 ist analog wie bei den Kundenattributen zu verfahren.

7.2 Erweiterung der Kartenanbieterdatenbank

Sobald das Schema der Kartenanbieterdatenbank geändert wird, muss überprüft werden, ob noch alle bisherigen Importe funktionieren oder ob diese angepasst werden müssen. Wird das Datenbankschema lediglich erweitert, so ist zu überprüfen, ob neue Daten eingefügt werden können, die bisher nicht kompatibel zum alten Schema waren.

7.3 Erweiterung um weitere Datenbanken

Die bisherige Integration hat sich mit der Integration der beiden Händlerdatenbanken in die Kartenanbieterdatenbank beschäftigt. Soll eine weitere Datenbank (Händler, Kartenanbieter) berücksichtigt werden, wird der bisherige Code nicht verändert, sondern erweitert.

7.3.1 Erweiterung um Quelldatenbanken

Die vorhandenen Klassen *ImportHaendler1* und *ImportHaendler2* behandeln die Datenintegration zwischen den jeweiligen Händlern und dem Kartenanbieter. Für jeden neuen Händler muss eine neue *ImportHaendlerX-Klasse* geschrieben werden. Es kann zwar grundsätzlich der gleiche Aufbau wie bei den bisherigen beiden Klassen verwendet werden, es existiert aber keine Oberklasse zu den *ImportHaendler-Klassen*, die einen Rahmen festsetzt. Eventuell sind neue Klassen für den initialen Import oder die Bereinigung der Ausgangsdaten zu implementieren.

7.3.2 Erweiterung um Zieldatenbanken

Wird das Integrationsszenario auf weitere Kartenanbieter ausgeweitet, so ist für jede Händlerdatenbank eine neue Klasse zu schreiben, da die bisherigen speziell auf den Import in die bekannte Kartenanbieterdatenbank ausgerichtet sind. Die Klassen *Loader* und *DBConnection* sind jedoch so ausgelegt, dass keine Anpassungen vorgenommen werden müssen. Eventuell sind neue Klassen für den initialen Import oder der Bereinigung der Ausgangsdaten zu entfernen.

7.3.3 Technische Erweiterbarkeit

Im Falle einer Erweiterung um ein weiteres Händlerdatenbankschema, das sich von den zwei bereits implementierten Schemata unterscheidet, müssen zwei Arbeitsschritte durchgeführt werden. Zunächst muss eine Importklasse implementiert werden, die alle notwendigen Schritte der Datenintegration analog zu den zwei bereits implementierten *ImportHändler*-Klassen durchführt. Schließlich muss die Klasse *ImportInitData* für den neu zu integrierenden Händler erweitert werden.

8 Probleme bei der Implementierung

Im Verlauf der Implementierung und Programmierung sind diverse unvorhergesehene Probleme aufgetreten, die im Folgenden einzeln erläutert und zu denen die von den Mitgliedern der Projektgruppe umgesetzten Lösungen vorgestellt werden.

8.1 Gemeinsame Probleme beim Import der Daten beider Händlerdatenbanken

Es hat Probleme gegeben, die sowohl beim Import der Daten aus der Datenbank des Händler 1 als auch beim Import der Daten aus der Datenbank des Händler 2 aufgetreten sind.
Hierzu gehört das Problem von Attributen, die zwar in den Händlerdatenbanken vorhanden gewesen sind, nicht jedoch in der Datenbank des Kartenanbieters. Dieses Problem wurde durch das Anlegen von im Kartenanbieterdatenbankschema dafür vorgesehenen dynamischen Attributen gelöst.
Ein weiteres Problem beider Händler stellte das Vorhandensein identischer Werte (beispielsweise IDs) in den beiden Händlerdatenbanken dar. Dieses Problem ist umgangen worden, indem eine HändlerID für jeden Händler in die entsprechenden Entitäten des Kartenanbieterdatenbankschemas eingeführt worden ist.
Weiterhin ist ein Problem aus der Verwendung von Sequenz-IDs in den Datenbankschemata der beiden Händler entstanden. Hierfür ist in der Kartenanbieterdatenbank kein Attribut vorgesehen, so dass unter der Annahme, dass Oracle die Daten für die *TZA*-Werte ausreichend genau behandelt, die Sequenz-IDs weggelassen werden können, da die Primärschlüssel-Kombinationen in diesem Fall bereits eindeutig sind.
Des Weiteren haben die Daten in den Händlerdatenbanken eine unzureichende Zeitgenauigkeit bzgl. der Zeitattribute wie z.B. *TZA*,

TZE, *GZA*, *GZE* aufgewiesen. Eine sauberere Lösung hätte die Verwendung von Daten, die aus der Funktion *SYSTIMESTAMP* generiert werden, anstelle von Daten, die aus der Funktion *SYSDATE* generiert werden, beim Erstellen der Testdaten in den Händler-Datenbankschemata dargestellt. Durch die zusätzliche Einführung von HändlerIDs ist die Verwendung von den ungenaueren *SYSDATE*-Daten jedoch kein gravierendes Problem.

Ein schwerwiegenderes Problem der Testdaten hat das Vorhandensein von Datensätzen mit identischer Artikelnummer innerhalb ein und derselben Transaktion dargestellt. Beim Import in die Kartenanbieterdatenbank wird aus den TransaktionsIDs und den Artikelnummern ein Primärschlüssel für die Entität *Posten* generiert, der bei doppeltem Vorkommen derselben Artikelnummer innerhalb einer Transaktion seine Eindeutigkeit verliert. Gelöst wurde dieses Problem, indem diese Datensätze aus den mit Testdaten gefüllten Händlerdatenbanken gelöscht wurden. Dies war in diesem Fall möglich, da beim Erstellen der Testdaten in den Händlerdatenbanken der Fehler begangen wurde, nicht fehlerfreie Testdaten zu erstellen. Bei der Implementierung wird davon ausgegangen, dass die Daten in den Händlerdatenbanken fehlerfrei vorliegen.

Außerdem hat ein großes Datenvolumen der Testdaten in den Händlerdatenbanken die Laufzeit der Importe verlängert. In den ersten Testphasen ist das Problem der langen Laufzeit umgangen worden, indem die Auswahl der zu importierenden Testdaten für die einzelnen Entitäten auf eine geringe Anzahl reduziert worden ist.

Aus programmiertechnischer Sicht haben Probleme mit Oracle-Locks und Schleifenverschachtelungen in Java bzgl. der Datenbankobjektzugriffe den Fortschritt des Programmierens aufgehalten.

Zusätzlich zu den bereits beschriebenen Problemen ist noch ein weiteres Problem aufgetreten. Einige initiale Daten müssen in der Kartenanbieterdatenbank angelegt werden, bevor der Import der Händler-Daten durchgeführt werden kann. So müssen beispielsweise Händler und Kartensysteme angelegt und dynamische Attribute erzeugt werden. Diese Aufgabe wird von der Klasse *importInitData* ausgeführt. Sie wird aufgerufen, indem der Klasse *Loader* der Parameter *-init* übergeben wird.

8.2 Probleme beim Import der Daten des Händler 1

Im Zusammenhang mit Händler 1 ist das Problem der Hierarchiebildung bezüglich verschiedener Warenkategorien aufgetreten, wobei während des Anlegens die Reihenfolge beachtet werden musste. Eine bessere Lösung dieses Problems würde die Erstellung von Baumstrukturen während der Programmlaufzeit darstellen. Aufgrund der Beschaffenheit der Testdaten bzgl. einer Ordnung in den Warengruppen ist eine Behandlung dieses Problems nicht erforderlich.

8.3 Probleme beim Import der Daten des Händler 2

Im Datenbankschema von Händler 2 stellt die Hausnummer zusammen mit der Straße ein Attribut dar, welches eine Verletzung der Atomarität darstellt. Im Kartenanbieterdatenbankschema hingegen bildet die Hausnummer ein eigenes Attribut. Dieses Problem ist durch die Einführung der Hausnummer *-1* gelöst worden. Eine Alternative stellt das Parsen des Attributs dar, das die Straße und die Hausnummer beinhaltet.
Ein weiteres Problem von Händler 2 stellen zudem fehlende Werte für Primärschlüssel im Datenbankschema des Kartenanbieters dar. Gelöst wurde dieses Problem durch das Anlegen von künstlichen Werten wie zum Beispiel den Wert *-1* für *NUMBER*-Attribute oder das Datum *01-Jan-1900* für *TIMESTAMP*-Attribute.
Des Weiteren fehlt im Datenbankschema von Händler 2 eine Relation zwischen den Entitäten *Shop* und *Produkt*, wobei diese im Datenbankschema des Kartenanbieters hingegen im Primärschlüssel enthalten ist. Um dieses Problem zu lösen, wurden für jede Filiale alle Produkte aus dem Datenbankschema von Händler 2 einzeln angelegt.
Das nächste Problem im Datenbankschema von Händler 2 stellt das fehlende Attribut *Staat* in der Entität *Filiale* dar. Die Lösung für dieses Problem stellt die Einführung des Strings *Deutschland* dar, wobei dabei vorausgesetzt werden muss, dass alle Kunden von Händler 2 aus Deutschland kommen.
Außerdem ist das Problem fehlender Transaktions- und Gültigkeitszeiten für die Entität Kategorie im Datenbankschema von Händler 2 aufgetreten. Im Datenbankschema des Kartenanbieters werden sie da-

gegen für den Import in die Entität Warengruppe als Primärschlüssel verwendet. Gelöst wurde dieses Problem durch die Einführung künstlicher Werte für das Attribut *Transaktionszeitanfang*.
Das letzte Problem in Bezug auf den Import der Daten vom Händler 2 stellen fehlende lov-Tabellen im Kartenanbieterdatenbankschema dar. In diesem Fall umfasst die Lösung das Auslesen der Beschreibungen der Attribute über deren Werte aus den lov-Tabellen. Dieses erzeugt jedoch zusätzliche Queries auf der Händler 2-Datenbank unter der Zuhilfenahme von zusätzlichen Resultsets, mit denen die Werte aus den lov-Tabellen des Datenbankschemas von Händler 2 ausgelesen werden und dann in das Kartenanbieterdatenbankschema integriert werden können.

9 Zusammenfassung

Die Aufgabe der Integration im Rahmen der Projektgruppe „Personalisierung internetbasierter Handelsszenarien“ ist es gewesen, die Daten aus den vorhandenen Händlerdatenbanken in das Kartenanbieterdatenbankschema zu integrieren. Die Umsetzung der Aufgabenstellung sowie dabei aufgetretene Probleme und deren Lösung wurden eingehend beschrieben. Auch wurde auf die Erweiterbarkeit der entstandenen Software eingegangen und eine detaillierte Klassenbeschreibung sowie ein Benutzerhandbuch zur Bedienung der Software erstellt und genauer betrachtet.

Teil III
Analyse

10 Anforderungsdefinition

Es soll ein Personalisierungs-Framework inklusive einer Data-Mining-Bibliothek erstellt werden, das verteilte Datenanalysen und im Speziellen auch temporales Data-Mining unterstützt. Dabei soll der Datenaustausch und die Steuerung des Analyseprozesses metadatengesteuert realisiert werden. Die Realisierung wird in mehreren Einzelschritten durchgeführt, um eine bessere Übersicht zu gewährleisten. Als Grundlage der zu implementierenden Bibliothek dient WEKA[WF01] als bereits etablierte Bibliothek für traditionelle Data-Mining-Anwendungen. Traditionell bedeutet in diesem Zusammenhang, dass WEKA bisher keine temporalen Analysen unterstützt.

10.1 Vorgehen in Teilschritten

Die Umsetzung der Gesamtanforderung erfolgt in mehreren Teilschritten, die jeweils in eigenen Anforderungsdefinitionen genau spezifiziert werden. In der vorliegenden Anforderungsdefinition wird daher nur der erste Teilschritt beschrieben. Das Analyseszenario also wird iterativ aufgebaut, so daß auf diese Weise Erfahrungen aus früheren Teilschritten einfließen können.
In diesem Teilschritt soll ein funktionierendes Framework und die grundlegende Möglichkeit zum temporalen Data Mining geschaffen werden. Als Datenquelle soll hier vorerst nur die Datenbank des Kartenanbieters verwendet werden. Dazu sind folgende Schritte notwendig:

- Implementierung einer rudimentären Import-/Exportmöglichkeit unter Vernachlässigung eines universellen Austauschformates
- Erweiterung WEKAs zur Verwaltung temporaler Daten
- Anpassung eines WEKA-Cluster-Algorithmus falls notwendig

- Implementierung eines Algorithmus zur Auffindung kalendarischer Muster
- Bereitstellung der Analyseergebnisse in einem Austauschformat

10.2 Systemeinsatz und -umgebung

Die Quelldatenbank befindet sich auf einem Oracle-Server (Version 9iR2). Da die zugrundeliegende WEKA-Bibliothek in Java implementiert ist, werden das Framework und die Erweiterungen ebenfalls in Java realisiert. Hierdurch soll ein möglichst plattformunabhängiger Einsatz des Frameworks gewährleistet werden.

10.3 Funktionale Anforderungen

In Abbildung 10.1 sind die Hauptbestandteile des zu implementierenden Frameworks dargestellt. Zunächst ist ein Datenstrom zu implementieren, der für den Import der Quelldaten sowie den Export der Ergebnisse zuständig ist. Der Strom soll die Möglichkeit bieten, über Metadaten gesteuert zu werden und große Datenmengen schrittweise verarbeiten zu können. Letzteres hat den Vorteil, Datenmengen einlesen zu können, die nicht vollständig in den Hauptspeicher passen.
Weiterhin soll es über Kontrolldaten möglich sein, das Framework zur Laufzeit konfigurieren zu können. Dies schafft die Voraussetzung, um wiederkehrende Analysen vorzukonfigurieren, Algorithmen auszuwählen und diese automatisiert ablaufen zu lassen.
Eine zentrale Rolle spielt die Erweiterung WEKAs um temporale Datenhaltung und Datenanalysen. Das Framework sorgt für die korrekte Ansteuerung der Bibliothek anhand der Kontrolldaten. Im Folgenden werden die funktionalen Anforderungen der einzelnen Teilbereiche für den ersten Schritt detailliert dargestellt.
Das Framework stellt kein eigenständiges Programm dar, sondern kann nur als Bibliothek in anderen Programmen verwendet werden. Dazu wird das Framework als Klasse instanziiert und bietet dann die Möglichkeit, Analysen anzustoßen. In der ersten, durch diesen Teilschritt realisierten Form, wird es nur möglich sein, die zwei zu implementierenden Algorithmen zu verwenden. Diese werden vorraussichtlich durch zwei spezielle Funktionen oder über einen Parameter gestartet. In einem späteren Schritt soll die Auswahl der Algorithmen

und der Analysedaten über Metadaten gesteuert werden. Die genaue Definition des Metadatenformates wird ebenfalls zu einem späteren Zeitpunkt definiert, da sie für diese Anforderungsdefintion nicht von Belang ist. Die Ergebnisse werden in einem einfachen, für die aktuell verwendeten Algorithmen spezalisierten Format gespeichert, das nach Möglichkeit sowohl menschen- als auch maschinenlesbar sein soll. Mit der Einführung der Metadaten für Datenauswahl und Steuerung soll auch das Ausgabeformat durch Metadaten festgelegt bzw. beschrieben werden.

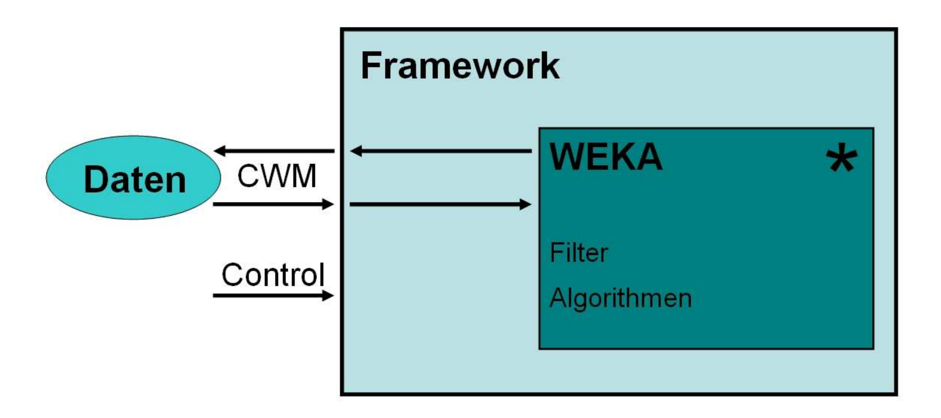


Abbildung 10.1: Schematische Darstellung des Frameworks

Datenstrom Für den ersten Teilschritt soll der Datenstrom keine Metadaten beachten. Konkret bedeutet dies für die Implementierung, dass insgesamt vier spezialisierte Datenströme programmiert werden müssen. Es ist jeweils ein Datenstrom zum Import für die beiden Algorithmen nötig. Da die beiden oben genannten Algorithmen unterschiedliche Daten benötigen, sind hier ebenfalls unterschiedliche Importströme nötig. Dasselbe gilt für den Export. Da die Ergebnisdaten von unterschiedlicher Form sein werden, müssen deshalb auch jeweils spezielle Ströme implementiert werden. Zusätzlich muß WEKA angepasst werden, damit eine Zusammenarbeit mit den Strömen möglich ist.

Importströme Die Importströme lesen die Daten direkt aus der Datenbank des Kartenanbieters. Je nach gewähltem Algorithmus wer-

den vordefinierte Daten aus der Datenbank eingelesen und WEKA zur Verfügung gestellt. Welche Daten dabei eingelesen werden, kann nicht konfiguriert werden. Diese Möglichkeit soll in einem späteren Teilschritt implementiert werden.
Der Datenstrom wird dabei so konzipiert, dass er als eine Art Cache funktioniert, d.h. es werden nicht alle verfügbaren Daten auf einmal gelesen, sondern es wird immer nur eine bestimmte Teilmenge in den Hauptspeicher geladen. Dies hat den Vorteil, dass zum einen der Hauptspeicher geschont wird und zum anderen die Zahl der Festplattenzugriffe optimiert wird.
Die API zur Kommunikation mit WEKA wird bereits jetzt so konzipiert, dass sie in Zukunft nicht mehr geändert werden muss, und somit in späteren Teilschritten nur noch die metadatenbasierte Steuerung des Stroms zu implemetieren bleibt.

Exportstöme Die Exportströme werden je nach eingesetztem Algorithmus die Ergebnisdaten in einem noch zu bestimmenden Format als textbasierte Datei ausgeben. Dabei wird das Format so gewählt, dass die Ergebnisdatei sowohl menschen- als auch maschinenlesbar ist.

Anpassung WEKAs Aktuell lädt WEKA die Daten mit Hilfe eines sogenannten Loaders und erzeugt aus den gelesenen Daten sogenannte Instanzen. Alle erzeugten Instanzen sind über eine Container-Klasse namens *Instances* zu erreichen. WEKA wird dahingehend angepasst, dass die *Instances*-Klasse bei Bedarf neue Daten aus dem Datenstrom nachlädt. Der Datenstrom übernimmt dabei die Verwaltung der Caching-Funktionalität.

10.3.1 Laufzeit-Konfiguration

In diesem Teilschritt wird keine Möglichkeit zur Konfiguration des Frameworks zur Laufzeit vorgesehen.

10.3.2 Temporale Erweiterung von WEKA

WEKA wird dahingehend erweitert, dass es zusätzlich zu den bisherigen Attributen auch temporale Attribute (Zeitstempel) speichern kann. Da diesen Attributen eine besondere Rolle zukommt, werden

sie als spezielle Attribute implementiert. Dieses ermöglicht eine einfache Verwendung derselben Daten sowohl zum normalen Data Mining als auch zum temporalen Data Mining, ohne dass Attribute zwischenzeitlich gefiltert werden müssen. Die Anpassung wird in Form einer Erweiterung der aktuellen *Instance* Klasse erreicht, indem von dieser Klasse geerbt wird. Dieses hat den Vorteil, dass die aktuell vorhandenen Algorithmen bzw. Klassen nicht oder evtl. nur minimal angepasst werden müssen.

10.3.3 Algorithmus für kalendarische Muster

Es wird ein Algorithmus zum Auffinden von kalendarischen Mustern [LNWJ01] implementiert. Dabei wird der Algorithmus so entworfen, dass er vorerst nur Muster der Form `<Jahr, Monat, Tag>` auffindet. Für einen späteren Teilschritt wird eine konfigurierbare Version geplant.

10.4 Nicht-Funktionale Anforderungen

Die verwendeten Algorithmen und Datenstrukturen werden so konzipiert, dass sie auch mit großen Datenmengen eingesetzt werden können. Ein Beispiel dafür ist die Implementierung des Stromes als Cache, um sowohl Hauptspeicher als auch Festplattenzugriffe zu optimieren. So sollte eine maximale Skalierbarkeit erreicht werden können. Die Software soll vorhersehbare Fehler weitestgehend abfangen und diese nach Möglichkeit ohne Programmabbruch behandeln. Ist ein Fortführen des Programmablaufes nicht möglich, so werden sinnvolle Fehlermeldungen ausgegeben.
Weitere nicht-funktionale Anforderungen sollen nicht erfüllt werden.

10.5 Benutzerschnittstellen

Nach außen sichtbar ist als API nur die Möglichkeit, einen Datenstrom anzugeben, der die Daten für den Algorithmus einliest. Der Algorithmus muss direkt aufgerufen werden, da noch keine automatische Konfiguration implementiert wird. Dazu wird die WEKA-API verwendet. Da sowohl die Import- als auch die Exportströme momentan speziell für die Kartenanbieter-Datenbank programmiert werden, bedarf es hier keiner näheren Spezifizierung des Formates der Daten.

10.6 Fehlerverhalten

Allgemein wird eine Fehlerbehandlung mit Hilfe von Exceptions realisiert. Dadurch kann der Benutzer des Frameworks eigenständig entscheiden, wie auftretende Fehler behandelt werden sollen. Fehler, die durch semantisch inkorrekte Daten entstehen, werden zur Laufzeit des Algorithmus ignoriert. Am Ende können die aufgetreten Fehler über eine Schnittstelle abgefragt und somit vom Benutzer behandelt werden. Dies soll bewirken, dass bei großen Datenmengen der Algorithmus nicht wegen möglicherweise unbedeutender Fehler abbricht. In Zukunft ist es denkbar, die Fehlerbehandlung ebenfalls konfigurierbar zu machen.

10.7 Dokumentationsanforderungen

Die Dokumentation des Programms umfasst folgende Punkte:

- Anforderungsdefinition
- Entwurf
- Dokumentierter Sourcecode
- Technisches Handbuch

Die Dokumentation soll in elektronischer Form erfolgen. Anforderungsdefinition und Entwurf werden in LaTeX verfasst, der Quellcode wird mit Javadoc kommentiert. Dadurch lässt sich aus den Kommentaren leicht eine API-Dokumentation in Form von HTML erstellen.

10.8 Abnahmekriterien

Bei der Abnahme sollten die oben definierten Anforderungen erfüllt sein. Die Algorithmen werden über zwei Anwendungsszenarien verifiziert.
Zur Verifizierung des Clustering-Algorithmus sollen die beim Kartenanbieter gespeicherten Kunden anhand der eingekauften Produkte geclustert werden. Die entstandenen Cluster könnten dann beispielsweise in einem Online-Shop zur Realisierung eines Recommender-Systems herangezogen werden. Ein solches System kann anhand der

Clusterzugehörigkeit einem Kunden alternative Produkte vorschlagen.
Der Algorithmus zum Auffinden kalendarischer Muster soll in den Warenkorb-Daten des Kartenanbieters temporale Muster finden. Diese Assoziationsregeln können benutzt werden, um abhängig vom Datum Produktempfehlungen auf Einstiegsseiten eines Online-Shops zu erzeugen oder um kalendarische Assoziationsregeln zu erzeugen, mit denen Kunden alternative Produkte oder etwa Produkt-Bundles angeboten werden können.

11 Entwurf

Wie in der Anforderungsdefinition (siehe Kapitel 10) vorgegeben, ist die Klassenbibliothek WEKA um temporale Datenhaltung, einen Algorithmus zum Auffinden kalenderbasierter Pattern und um eine Caching-Funktionalität zu erweitern. Die temporale Datenhaltung soll es WEKA ermöglichen, temporale Datenanalysen auszuführen, ohne dass die vorhandenen nicht-temporalen Algorithmen angepasst werden müssen. Die Caching-Funktionalität hat die Aufgabe, den Hauptspeicherverbrauch WEKAs zu reduzieren. Die aktuelle Implementierung lädt alle zu analysierenden Daten in den Hauptspeicher, was gerade bei großen Mengen von Daten zu Problemen führen kann. Der Cache soll immer nur einen Teil der Daten im Hauptspeicher halten, und zusätzlich die Daten, für wiederholte Iterationen, lokal zwischenspeichern. Dadurch soll eine Verminderung des Netzwerkverkehrs erreicht werden.

11.1 Architektur

Ein wichtiger Punkt bei der Implementierung der zusätzlichen Funktionen ist die Erhaltung der aktuellen WEKA-APIs. Dadurch sollen vorhandene Programme, die WEKA nutzen, auch weiterhin ohne Anpassungen funktionieren. Außerdem sollen innerhalb WEKAs so wenig Anpassungen wie möglich nötig sein. Die Architektur ist daher hauptsächlich durch die aktuelle WEKA-Architektur gegeben.
An dieser Stelle wird kurz die aktuelle WEKA-Architektur vorgestellt. Anschließend wird darauf eingegangen, wie die Erweiterungen in diese Architektur integriert werden.

11.1.1 WEKA-Architektur

WEKA stellt nicht nur Data-Mining-Algorithmen zur Verfügung, sondern auch ein Framework zur Verwaltung von Instanzen, inklusive Funktionalitäten zum Modifizieren der Attribute und der Reihenfolge der Daten . Ein zentrales Element stellt dabei die Klasse *Instances*

dar, die die Verwaltung der Instanzen übernimmt. Diese Klasse benutzt sogenannte Loader, um die Daten aus einer Datenquelle (i.d.R. ARFF-Dateien) zu laden. Wie bereits erwähnt, werden alle Daten im Hauptspeicher gehalten. Die *Instances*-Klasse verwaltet dabei die Meta-Informationen über die Attribute der Instanzen. Eine Instanz (gekapselt in der Klassen *Instance*) hat selber keine Informationen darüber, welche Attribute sie besitzt. Die *Instances*-Klasse weist jedem Attribut einen Index zu und speichert die Attributwerte in einer Instanz nur anhand dieses Indexes. Um diesen Index anhand des Attributnamens zu bestimmen, erhält jedes *Instance*-Objekt eine Referenz auf das verwaltende *Instances*-Objekt.
Die bereits in WEKA vorhandenen Algorithmen greifen auf die Daten per Index zu. Sie erhalten dazu eine Referenz auf die *Instances*-Klasse und können so auf jedes beliebige Element zugreifen.
Temporale Daten können in WEKA momentan nicht explizit gespeichert werden. Eine Speicherung der temporalen Informationen über Standardattribute wäre möglich, jedoch ist es i.d.R. erstrebenswert, zwischen temporalen Attributen und „normalen“ Attributen zu unterscheiden.
Da der Algorithmus zum Auffinden kalendarischer Muster vorher nicht vorhanden war, musste auf keine Konventionen von WEKA Rücksicht genommen werden. Der Zugriff auf den Algorithmus soll aber weitestgehend ähnlich der WEKA-Algorithmen stattfinden.
TemporalAssociationRules ist eine eigenständige Klasse, welche auf die angepassten Klassen *Instance*, *Instances* und *Attribute* zurückgreift.

11.1.2 Design der WEKA-Erweiterungen

Zentral für die Erweiterung ist die temporale Datenhaltung. Dazu wurde eine neue Klasse entwickelt, die temporale Informationen speichern kann. Die Klasse erbt von der vorhandenen Klasse *Instance* und wurde *TemporalInstance* genannt. Die temporalen Informationen werden als zusätzliche Felder vom Typ *Timestamp* gespeichert. Durch die Vererbung können Instanzen vom Typ *TemporalInstance* auch in nicht-temporalen Algorithmen verwendet werden.
Bei dem Entwurf des Caches wurde eine Neuimplementierung der *Instances*-Klasse gewählt. Die aktuelle API wird dabei komplett übernommen und zusätzlich erweitert. Dadurch ist für neue Algorithmen eine intuitivere API gegeben, ohne dass die alten Algorithmen an-

gepasst werden müssen. Intern wird die Klasse jedoch komplett neu implementiert.
Zur Umsetzung des Cachings wird die *Instances*-Klasse in drei einzelne Klassen aufgeteilt. Da ein Großteil der bekannten Algorithmen iterativ auf die Daten zugreift, ist es das Ziel, diesen iterativen Zugriff möglichst performant zu machen. Als Anbindung zur Datenquelle wurde daher eine neue eigene Klasse entworfen, die ein Forward-Iterator-Interface (siehe Abschnitt 12.1) bietet. Als Interface dient *SourceIterator*. Für den Datenbankzugriff, der im Kontext des zu Grunde liegenden Szenarios benötigt wird, wurde eine abstrakte Klasse *JDBCSourceIterator* entwickelt, die diejenigen Methoden des *SourceIterator*-Interfaces implementiert, die unabhängig von den zu liefernden Daten sind. In einem späteren Schritt wird die abstrakte Klasse *JDBCSourceIterator* Grundlage für datenbankbasierte metadatengesteuerte Zugriffe auf Datenquellen sein. Der Source-Iterator ist somit die direkte Anbindung an die Datenquelle.
Zur Implementierung des Cachings dient die Klasse *InstancesCache*, die über den Source-Iterator auf die Daten zugreift. Die Klasse lädt immer eine bestimmte (frei zu konfigurierende) Anzahl von Instanzen in den Hauptspeicher und stellt diese in Form eines Forward-Iterator-Interfaces und eines Random-Access-Interfaces zur Verfügung. Per Option kann die Klasse dazu veranlasst werden, die bereits geladenen Daten lokal in einem sogenannten Swap zwischenzuspeichern. Gerade bei Datenquellen, auf die per Netzwerk zugegriffen werden muss, kann dies die Performanz verbessern. Aber auch generell entfällt so später das Verarbeiten der Metadaten bei wiederholten Zugriffen und es werden evtl. notwendige Joins verhindert. Zur Zwischenspeicherung der Daten wird die als Open Source frei verfügbare Berkeley DB [Sof03] verwendet.
Die dritte Klasse ist die Klasse *Instances* selber. Die API bleibt, wie bereits erwähnt, komplett erhalten. Jedoch werden alle Funktionen zum Zugriff auf die Daten an die darunter liegende Klasse *InstanceCache* weitergeleitet. Die *Instances*-Klasse selber übernimmt damit nur noch Funktionen zur Verwaltung und Manipulation der Attribute.
Der Algorithmus zum Auffinden kalendarischer Muster wurde in drei Klassen realisiert. Der Algorithmus ist in der Klasse *TemporalAssociationRules* implementiert, die vom Framework instantiiert und ausgeführt wird. Dabei muss eine Menge von Instanzen, welche die Positionen von Einkäufen darstellen, sowie ein Beobachtungsinter-

vall mittels Start- und Enddatum übergeben werden. Des Weiteren muss ein Mindest-Support angegeben werden, der bestimmt, wie hoch das prozentuale Aufkommen eines Artikels für einen Tag ist, um als *häufig* zu gelten. Dem Algorithmus kann als optionales Argument noch ein *Fuzzy-Support* übergeben werden. Fehlt dieser Wert, wird als Default ein Wert von 1 angenommen (siehe auch Kapitel 13, sowie Abschnitt 14.3.3).
Zum Speichern einzelner Warenkörbe inklusive erweiterter Informationen, wird die Klasse *Basket* modelliert. Eine Menge von Warenkörben soll in der Klasse *SetOfBaskets* dargestellt werden, die auch angepasste Funktionen für *TemporalAssociationRules* bereithält. Die Analyse-Ergebnisse werden ebenfalls in Form von *SetOfBaskets* repräsentiert, werden aber durch Aufruf der Funktion *getResults()* in ein Objekt der Klasse *Instances* transformiert.

11.2 Klassenbeschreibung

11.2.1 TemporalInstance

Die API der Klasse *Instance* bleibt komplett erhalten. Zusätzlich dazu werden folgende Funktionen zum Zugriff auf die temporalen Daten implementiert:

- *get/set*-Methoden zum Zugriff auf die Zeitstempel für Transaktionszeit-Anfang und -Ende, sowie Gültigkeitszeit-Anfang und -Ende, abgekürzt: *tza, tze, gza, gze*. Die Zeitstempel sind als *Timestamp*s gespeichert.

11.2.2 SourceIterator

Das Interface *SourceIterator* stellt das Basisinterface für alle Klassen dar, die den Zugriff auf eine Datenquelle implementieren sollen. Es definiert die folgenden grundlegenden Methoden, die ein SourceIterator mindestens bieten muss:

- *boolean hasNext()* Gibt zurück, ob noch weitere Datensätze geliefert werden können.

- *Instance next()* Gibt den nächsten Datensatz zurück, wenn dieser existiert. Wirft eine *NoSuchElementException*, falls keine Datensätze mehr geliefert werden können.

- *void reset()* Initialisiert den Iterator neu, so dass der nächste Datensatz wieder der erste Datensatz der Datenquelle ist.

11.2.3 JDBCSourceIterator, ClusteringInstance und CalendarInstance

Diese Klassen implementieren das Interface *SourceIterator*. *JDBC-SourceIterator* ist dabei eine abstrakte Basisklasse für den Zugriff auf SQL-basierte Datenbanken und implementiert alle datenunabhängigen Funktionen. Die verbleibenden zwei Klassen liefern spezialisierte Daten zurück, die für jeweils einen Algorithmus angepasst sind.

11.2.4 InstanceCache

- *InstanceCache(SourceIterator Source, boolean swap_enabled, int size, String database)* Konstruktor der Klasse. Die Parameter haben folgende Bedeutung:
  - *Source* Referenz auf den *SourceIterator*, der den Zugriff auf die Datenquelle bereit stellt.
  - *swap_enabled* Hierdurch kann das lokale Zwischenspeichern der Daten aktiviert oder deaktiviert werden.
  - *size* Größe des Caches in Anzahl an Datensätzen.
  - *database* Name der Berkeley DB, die als lokaler Swap fungiert. Dieser Parameter ist optional.
- *void close()* Signalisiert dem Cache, dass die Swap-DB geschlossen werden kann.
- *Instance getElementAt(int)* Liefert das Element an der angegebenen Position zurück.
- *boolean hasNext()* Liefert *true*, wenn noch ein weiteres Element zurückgeliefert werden kann. Der Rückgabewert hängt nicht davon ab, ob noch ein Element aus der Datenquelle geholt werden kann, sondern ob ein weiteres Element nach dem letzten gelieferten Element existiert.
- *Instance next()* Liefert das nächste Element zurück, wenn es existiert. Wirft ansonsten eine Exception.
- *int size()* Liefert die Größe des Caches zurück.

11.2.5 Instances

Die API dieser Klasse bleibt bis auf zwei zusätzliche Funktionen unverändert. Dadurch ist sichergestellt, dass die vorhandenen Algorithmen und Programme auch weiterhin problemlos funktionieren, aber die Vorteile der Neuimplementierung nutzen können. Zusätzlich implementiert wurden die Funktionen

- *boolean hasNext()* Liefert zurück, ob ein weiteres Element zurückgeliefert werden kann. Hier gilt dasselbe wie bei *InstanceCache.hasNext()*.

- *Instance next()* Liefert die nächste Instanz zurück, wenn diese existiert. Wirft ansonsten eine Exception. Für zukünftige Algorithmen wird diese Funktion empfohlen, da der Overhead eines Random-Access-Zugriffs entfällt.

11.2.6 TemporalAssociationRules

Diese Klasse implementiert den Algorithmus zum Auffinden kalendarischer Muster. Er wird mittels folgender Funktionen gesteuert:

- *TemporalAssociationRules(Date min, Date max, float minsupp)* Konstruktor der Klasse. Die Parameter haben die folgende Bedeutung:

  - *min* Gibt ein Datum an, ab welchem der Algorithmus Datensätze zur Analyse beachten soll. Das Datum ist inklusiv.
  - *max* Gibt ein Datum an, bis zu welchem der Algorithmus Datensätze auswerten soll. Das Datum ist exklusiv.
  - *minsupp* Gibt den Mindest-Support für den Algorithmus an, welcher ausdrückt, wie oft eine Artikelkombination prozentual an einem Tag vorkommt, um als häufig zu gelten. Der Wert muss zwischen 0 und 1 liegen.
  - *fuzzy* Gibt den Mindest-Support an, für den häufige Kombinationen in einem Wildcard-Muster (auch Star-Pattern genannt, siehe Abschnitt 13, sowie 17.1) prozentual vorkommen müssen. Der Wert muss zwischen 0 und 1 liegen.

- *buildAssociations(Instances transactions)* Mit dieser Methode wird die eigentliche Analyse gestartet. Sie startet die einzelnen Analyseschritte und produziert die Ergebnisse der kalendarischen Mustersuche.
  - *transactions* Hier müssen die Transaktionen übergeben werden, aus denen die zu untersuchenden Datensätze entnommen werden sollen. Dabei müssen die Instanzen nach der Zeit aufsteigend sortiert sein und die *KundenID*s verschiedener Warenkörbe tagweise disjunkt sein. Das bedeutet, dass zwei aufeinanderfolgende Warenkörbe mit der gleichen ID nicht am selben Tag auftreten dürfen.
- *instances getResults* Liefert die gefunden Ergebnisse als Instanzen zurück. Diese beinhalten die 6 Attribute Jahr, Monat, Tag, den Support-Wert, eine eindeutige Regel-ID und einen Vektor, der die Artikel beinhaltet, die für dieses Datum häufig waren. Mit dem Wert -1 für die Position Jahr, Monat oder Tag wird angezeigt, dass es sich um ein Wildcard-Muster für diese Position handelt.

11.2.7 Basket

Diese Klasse modelliert einen Warenkorb mittels eines *FastVector*s für die Artikel und eines *GregorianCalendar*s für den Tag des Einkaufs. Für den Algorithmus wurden noch drei Zähler eingefügt, die vom Algorithmus verwendet werden, um die Anzahl der Gesamttransaktionen, der Häufgkeit des aktuellen Warenkorbs und den zugehörigen Fuzzy-Match für einen Tag abzuspeichern. Die Klasse stellt folgende Funktionen bereit:

- *get/set*-Methoden für die einzelnen Zähler und das Datum.
- *equals*-Methoden für gesamte Warenkörbe, für die Zeitstempel von Warenkörben und den Inhalt von Warenkörben.
- Methoden zum Hinzufügen von Artikeln zu Warenkörben.
- eine Methode zur Bildung der Vereinigungsmenge zweier Warenkörbe, wenn diese sich nur in einem Artikel unterscheiden.

11.2.8 SetOfBaskets

Diese Klasse bietet Methoden zur Verwaltung einer Menge von Warenkörben. Ein *SetOfBaskets* speichert eine Menge von *Basket*s in einem *FastVector* zusammen mit der Gesamtanzahl der Transaktionen für einen Tag. Die Anzahl der Warenkörbe lässt sich über die Länge des *FastVector*s feststellen. Es werden folgende Methoden bereitgestellt:

- *get/set*-Methoden Methoden für den Zähler.
- Methoden zum Hinzufügen und Entfernen von Warenkörben.
- *fuzzyUpdate* zum automatischen Inkrementieren des Fuzzy-Zählers für alle Warenkörbe des aktuellen *SetOfBaskets*.
- *MixToNPlus1Sets* Bildet aus allen Warenkörben, wobei alle die gleiche Größe N haben, neue Warenkörbe der Größe $N + 1$, wenn sich die einzelnen Warenkörbe nur in einem Artikel unterscheiden.

12 Technisches Handbuch: Cache

Im Folgenden soll die technische Realisierung des im Entwurf (siehe Kapitel 11) erwähnten Caches beschrieben werden. Der Cache trägt dafür Sorge, dass immer nur ein begrenzter Teil der Daten im Hauptspeicher vorgehalten wird. Die Daten liest er selber immer aus einem *SourceIterator* und speichert sie lokal in einer Berkeley DB [Sof03], die als Swap fungiert. Die Klasse *Instances* behält ihr Interface, damit möglichst wenige Änderungen an vorhandenem Code durchgeführt werden müssen. Allerdings werden jetzt alle Datenzugriffe direkt an den Cache weitergeleitet, d.h. die Klasse *Instances* dient nur noch als Metadaten-Container.

12.1 Klassendesign

Die drei zentralen Klassen, die für die Caching-Funktionalität sorgen, sind *Instances*, *InstanceCache* und *SourceIterator*. Die Abhängigkeiten der Klassen sind in Abbildung 12.1 zu sehen und im Folgenden beschrieben.
Die Klasse *Instances* hat ihr aus WEKA bekanntes Interface behalten [Dik03]. Zum einen hatte sie die Aufgabe, die Metadaten, im Speziellen also die Attributinformationen, zu verwalten, und zum anderen den Zugriff auf einzelne Instanzen zu liefern. Diese Funktionalität wurde aufgeteilt. Die Klasse *Instances* verwaltet jetzt nur noch die Attribut-Informationen und delegiert alle Datenzugriffe an die Klasse *InstanceCache*. Dazu erzeugt die Klasse *Instances* ein Objekt der Klasse *InstanceCache* und speichert eine Referenz darauf. Soll die Klasse *Instances* Daten aus einer Datenquelle lesen, so muss ihr eine Referenz auf ein Objekt der Klasse *SourceIterator* übergeben werden, die direkt an die Instanz der Klasse *InstanceCache* weitergereicht wird. Wird kein *SourceIterator* angegeben, müssen die Daten manuell der Instanz der Klasse *Instances* hinzugefügt werden.
Der *InstanceCache* implementiert zwei Aufgaben, nämlich das Laden der Daten aus dem *SourceIterator* in den Cache im Hauptspeicher

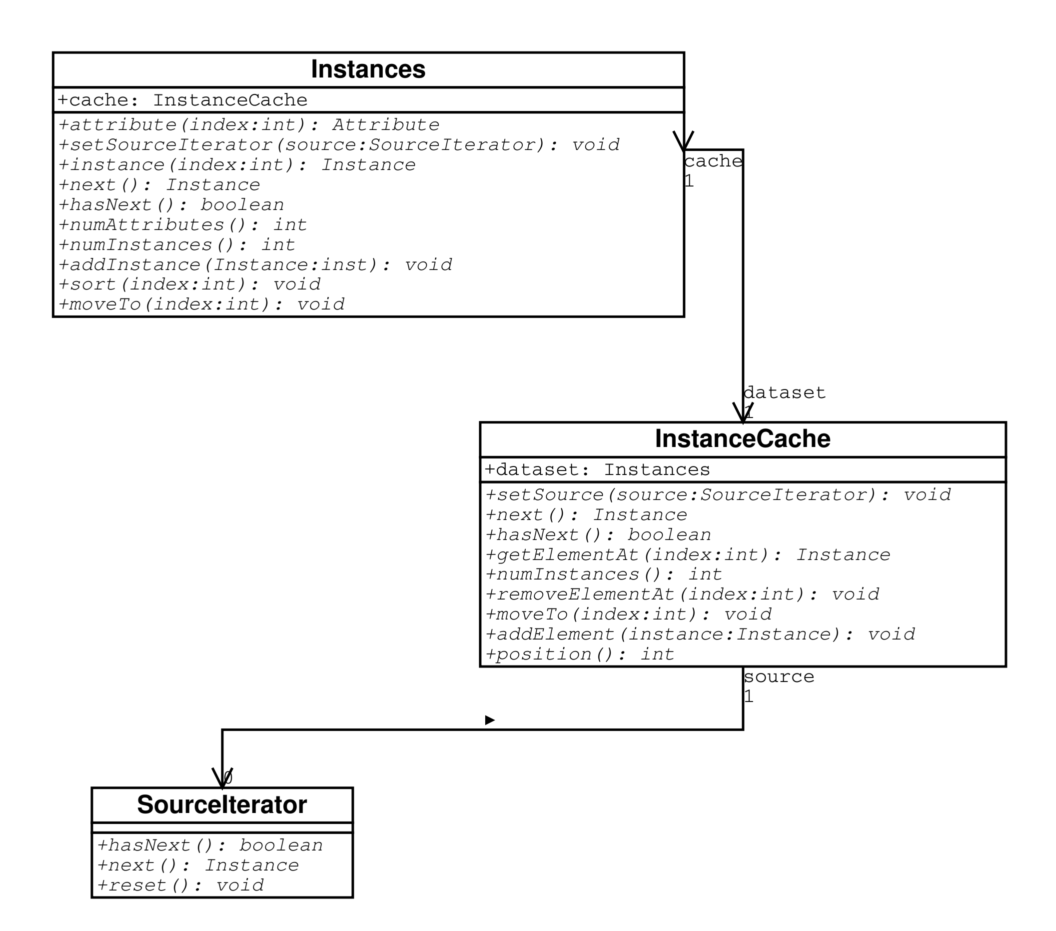


Abbildung 12.1: Klassendiagramm

und das Speichern der Daten im Swap. Letzteres wurde implementiert, damit bei mehrfachen Iterationen nicht immer wieder auf die eigentliche Datenquelle zugegriffen werden muss. Gerade bei einer Datenbank als Quelle und einem eher komplexen Statement, kann dies einen deutlichen Performanz-Verlust bedeuten. Im Swap werden die Daten als serialisierte Objekte gespeichert Um selber Zugriff auf die Metadaten der Instanzen zu haben, besitzt der *InstanceCache* eine Referenz auf die übergeordnete *Instances*. Diese beiden Klassen sind also doppelt verlinkt.
Der *SourceIterator* stellt die Anbindung zur Datenquelle dar. Sein Interface ist als sogenannter Forward-Iterator implementiert. Ein Forward-Iterator stellt eine API zur Verfügung, mit der auf Daten sequentiell zugegriffen werden kann. Die API besteht aus den zwei Funktionen *hasNext()* zum Überprüfen, ob noch ein weiteres Element im Iterator existiert, und *next()*, um das nächste Element anzufordern. Der Zugriff erfolgt somit rein sequentiell, und eine erneute Anforderung eines Elementes wird alleine vom *InstanceCache* bearbeitet.

12.2 Implementierungsdetails

12.2.1 Änderungen an der Klasse Instances

Intern bestehen die Änderungen daraus, dass Datenzugriffsmethoden als Wrapper für den Cache dienen, und alle Aufrufe direkt weiterleiten. Das Original-Interface ist erhalten geblieben, allerdings sind Methoden der *Forward-Iterator*-API hinzugekommen (siehe Abschnitt 12.1).

12.2.2 InstanceCache

Der Cache kann entweder mit einer Datenquelle initialisiert oder auch komplett zur Laufzeit generiert werden. Wird eine Datenquelle benutzt, lädt der Cache anfangs erstmal so viele Daten aus der Quelle, wie in den Hauptspeicher-Cache passen.[1] Die Größe des Cache kann per Parameter konfiguriert werden. Alle Zugriffe auf Elemente innerhalb des Cache werden direkt beantwortet. Der Zugriff kann entweder

[1] Mit Hauptspeicher-Cache, oder einfach nur Cache, ist immer die Datenstruktur im Hauptspeicher gemeint. Hingegen wird die Berkeley DB, die die Daten auf Platte zwischenspeichert, als Swap bezeichnet.

per Index oder durch Aufruf der Methode `next()` geschehen. Wird ein Element angefordert, das nicht im Cache liegt, gibt es zwei Fälle zu beachten. Entweder müssen Daten aus der Datenquelle nachgeladen werden oder die Daten wurden bereits aus der Datenquelle gelesen und liegen jetzt im Swap.
Das Nachladen der Daten geschieht der Einfachheit halber immer nach dem gleichen Schema: Es werden immer nur komplette Blöcke gelesen, die genau so groß wie der Cache selber sind. Ein solcher Block wird als *Seite* bezeichnet und enthält jeweils die konfigurierte Anzahl von Instanzen. Aus der Quelle kann aufgrund des *SourceIterator*-Interfaces nur vorwärts gelesen werden. Daher werden in dem Fall, dass per Index auf ein Element zugegriffen wird, das weiter hinten im Datenstrom liegt, alle dazwischen liegenden Seiten erst geladen und dann in den Swap geschrieben. Je nachdem wo die nächste Seite liegt, wird diese entweder aus der Quelle oder aus dem Swap geladen.
Im Swap sind die Elemente nach Index indiziert. Das bedeutet, das auch Zugriff per Index (Random Access) performant sein sollte. Werden Elemente gelöscht oder hinzugefügt, hat das immer erst ein komplettes Lesen aller Quelldaten zur Folge.
Zusammenfassend bedeutet dies, dass der Cache immer seitenweise liest. Sollte also ein Algorithmus häufig einzelne Datensätze an unterschiedlichen Stellen lesen, ist eine kleine Seitengröße[2] angebracht. Jedes Element wird nur einmal aus der Quelle gelesen, anschließend befindet es sich im Swap und wird von dort aus gelesen. Änderungen an den Daten bedeuten immer das komplette Lesen und Swappen[3] aller Daten der Quelle.

12.3 Fazit

Da das Hauptaugenmerk der Projektgruppe nicht beim Cache liegt, und der Cache nur die Benutzung WEKAs mit großen Datenmengen ermöglichen sollte, sind die verwendeten Paging[4]- und Swapping-Algorithmen sicherlich eher als rudimentär anzusehen. Gerade das komplette Lesen einer Seite ist nicht immer ideal, genauso kann es ungeschickt sein, beim Hinzufügen von Elementen immer alle Daten zu lesen.

[2]Größe einer einzelnen Seite im Cache.
[3]Lokales Zwischenspeichern der Daten
[4]Art und Weise, in der die Daten nachgeladen werden

Allerdings hängt der ideale Swapping-Algorithmus auch stark vom verwendeten Analysealgorithmus ab, der die Daten benötigt. So wird die *Instances-Klasse* in WEKA z.B. ebenfalls benutzt, um die Blätter eines Baums zu kapseln. Bei großen Bäumen entstehen jedoch so viele Blätter, dass die Anzahl der File-Deskriptoren des Betriebssystem nicht mehr ausreicht, um alle Berkeley DB Files zu verwalten.
Der Vorschlag der Projektgruppe ist daher, in Zukunft den Cache als Interface zu implementieren und verschiedene konkrete Implementierungen anzubieten, die mit unterschiedlichen Paging- und Swapping-Algorithmen arbeiten, oder vielleicht die Daten auch nur im Hauptspeicher halten. Dies würde jedoch auch eine Änderung der vorhandenen WEKA-API bedeuten, was offensichtlich grundlegende Änderungen an WEKA nach sich ziehen würde. Aus diesem Grunde wurde auf die Realisierung dieser Änderungen im Rahmen der Projektgruppe verzichtet.
Es wäre ebenfalls möglich, den *SourceIterator* dahingehend zu erweitern, dass er nicht nur einen rein sequentiellen Zugriff auf die Daten zulässt, sondern ein erweitertes Interface anbietet. Denkbar wäre ein Zugriff per Index, sequentieller Zugriff in beide Richtungen oder die Möglichkeit, Daten direkt in der Quelle zu ändern. Da die Algorithmen, die im Rahmen der PG verwendet werden, jedoch diese Möglichkeiten nicht benötigen, wurde auf die Implementierung verzichtet.

13 Technisches Handbuch: Algorithmus zum Auffinden kalendarischer Muster

Im Folgenden wird die technische Realisierung des im Entwurf erwähnten Algorithmus zum Auffinden kalendarischer Muster beschrieben. Der Algorithmus ist eine Umsetzung der in dem Artikel *„Discovering Calendar-based Temporal Association Rules“* [LNWJ01] beschriebenen Konzepte. Für eine Beschreibung des Alogrithmus und seiner Einsatzmöglichkeiten siehe Abschnitt 17.1.
Der Algortihmus benutzt den *Temporal-Apriori* Ansatz, um die Ausführungsgeschwindigkeit zu erhöhen und bietet weiterhin die Funktionalität des *Precise-* und des *Fuzzy-Matches*. Die Klasse *TemporalAssociationRules* führt die Berechnungen durch und benutzt zur Datenverwaltung die Klassen *Basket* und *SetOfBaskets*.

13.1 Funktionsweise des Algorithmus

Der Algorithmus zum Auffinden kalendarischer Muster (im Folgenden nur noch *der Algorithmus* genannt) soll aus einer Menge von Daten, welche alle mit einem Zeitpunkt versehen sind, Assoziationsregeln finden, die in einem vorgegebenen Beobachtungsintervall für elementare Zeitintervalle gelten. Dabei muss für elementare Zeitintervalle die Anforderung gelten, dass sie kompatibel zu den Zeitpunkten der Daten sind; das bedeutet, dass insbesondere die Länge der Zeitpunkte kleiner als die elementaren Zeitintervalle sein muss. Der Algorithmus arbeitet auf kalendarischen Mustern, welche die Form `<Jahr, Monat, Tag>` haben. Damit hat ein elementares Zeitintervall die Größe eines Tages.
Bei Assoziationsregeln wird des Weiteren einen Wert für den Mindest-Support benötigt, der angibt, ab welchem prozentualen Anteil eine Regel als häufig gilt. Bei dem Algorithmus wird dabei immer der prozentuale Anteil für ein elementares Zeitintervall betrachtet.

Der Algorithmus sucht nicht nur nach Assoziationsregeln für elementare Zeitintervalle, sondern untersucht die gefundenen Regeln zusätzlich auf eine prozentuale Häufigkeit in dem vorgegebenen Beobachtungsintervall. Die Häufigkeit ist in Bezug zu der Anzahl der elementaren Zeitintervalle zu verstehen. Wenn eine Assoziationsregel an jedem elementaren Zeitintervall innerhalb des Beobachtungsintervall gilt, wird von Precise-Match gesprochen. Dieses entspricht einem Fuzzy-Support-Wert von 100%. Der Fuzzy-Support-Wert gibt die Häufigkeit der Regeln in Bezug zu der Anzahl der elementaren Zeitintervalle im Beobachtungsintervall an.
Bei kalendarischen Mustern können zusätzlich Wildcards für die Zeiten angegeben werden, welche die Bedeutung von *für jeden* haben. Es werden also somit auch Assoziationsregeln für kalendarische Muster der Form *für jeden Tag*, *für jeden Monat* oder *für jedes Jahr* im Beobachtungsintervall gefunden. Auch sind Kombinationen möglich, wie zum Beispiel „für jeden Tag im November eines jeden Jahres“ im Beobachtungsintervall.
Wird der Fuzzy-Match zusammen mit einem Wildcard-Muster benutzt, berechnet sich der Fuzzy-Support aus der Anzahl der dem Muster zugehörigen elementaren Zeitintervalle.
Der Algorithmus erwartet Daten als Instanzen der Form „Kunden, Transaktions- oder Warenkorbnummer, Datum und Artikelnummer“, wobei die Instanzen nach dem Datum aufsteigend sortiert sein müssen und an einem Tag keine Kundennummer in zwei Warenkörben gleichzeitig vorkommen darf. Des Weiteren dürfen in einem Warenkorb die Artikelnummern nur disjunkt vorkommen.
Das Start- und Enddatum wird als *calendar* angegeben und die Support-Werte als *double*. Wird kein Fuzzy-Support-Wert angegeben, wird ein Default Wert von 1 angenommen, welches dem Precise-Match entspricht. Als Ergebnisse werden Instanzen erzeugt, die vom Framework in eine Datenbank gespeichert oder anders weiterverarbeitet werden können.

13.2 Klassendesign

Die drei Klassen, welche den Algorithmus implementieren, sind *TemporalAssociationRules*, *Basket* und *SetOfBaskets*. Zur Fehlerbehandlung in *TemporalAssociationRules* wurde zudem die Fehlerklasse *InvalidSupportException* implementiert. Die Abhängigkeiten der Klas-

sen werden im Folgenden beschrieben.
Die Klasse *TemporalAssociationRules* implementiert die Logik des Algorithmus. Sämtliche Berechnungen, sofern sie nicht einem bestimmten Datentyp zugeordnet sind, werden durch sie durchgeführt.
Zur Verwaltung der kalendarischen Muster und ihrer zugehörigen Assoziationsregeln wird ein dreidimensionales Array von *SetOfBaskets* angelegt, welches für jedes elementare Zeitintervall im Beobachtungsintervall (demnach für jeden Tag) ein *SetOfBaskets* enthält. In diesem werden alle Regeln, die für einen Tag oder ein Wildcard-Muster im Beobachtungsintervall halten, in Form von *Basket*s gespeichert. Dabei haben alle Baskets die gleiche Anzahl an Artikeln. Ein Basket repräsentiert dabei eine Assoziationsregel, die an dem zugehörigen Datum den Mindest-Support hält. Dieses geschieht, indem die Artikelnummern, die in einer Regel assoziiert werden, in einen Basket aufgenommen werden.
Alle gültigen Assoziationsregeln eines Kalendermusters werden in einem *SetOfBaskets* gehalten. Um Berechnungen zu erleichtern und zu beschleunigen, wird in den *SetOfBaskets* und den *Baskets* zusätzlich die Gesamtanzahl von Warenkörben für ein elementares Zeitintervall oder Wildcard-Muster abgespeichert. Die Klassen *SetOfBaskets* und *Basket* stellen für die Berechnung, die im Entwurf in Abschnitt 11.2 beschriebenen Funktionalitäten bereit.

13.3 Implementierungsdetails

Nachfolgend werden die Implementierungsdetails der einzelnen Klassen vorgestellt. Als weiterführende und ergänzende Informationen sei auf die Javadoc Dokumentation [Dik03] des Package *diko.weka.associations* verwiesen.

13.3.1 TemporalAssociationRules

Die Klasse *TemporalAssociationRules* führt die Logik des Algorithmusses aus. Dieses geschieht in mehreren Schritten. Der Aufruf der Klasse erfolgt mit den Parametern Mindest-Support-Wert, Start- und Enddatum sowie dem Fuzzy-Support-Wert. Falls der Fuzzy-Support-Wert nicht angegeben wird, wird er auf 1 gesetzt. Des Weiteren werden die übergebenen Werte auf ihre Plausibilität geprüft. Liegen die Werte für den Mindest-Support oder den Fuzzy-Support nicht zwi-

schen 0 und 1, wird eine *InvalidSupportException* geworfen und der Algorithmus beendet.
Wird ein späteres Start- als Enddatum angegeben, führt der Algorithmus die Berechnungen auf allen Instanzen durch, die ihm übergeben werden.
Der Konstruktor führt noch keine Berechnungen durch. Diese werden durch *buildAssociations* gestartet, wobei noch die zu untersuchenden Instanzen übergeben werden. *buildAssociations* stellt zunächst fest, welchen Start- und Endindices das Beobachtungsintervall innerhalb der übergebenen Instanzen hat. Danach wird das dreidimensionale Array zur Speicherung der Assoziationsregeln angelegt. Es hat die Form `<Jahr, Monat, Tag>` und die Größe bestimmt sich bei den Jahren aus der Differenz zwischen dem kleinsten und dem grössten Jahr zuzüglich einem Eintrag für die Wildcard-Muster. Bei Monat und Tag werden aufgrund der einfacheren Abfragen einfach jeweils 13 bzw. 32 Felder angelegt. Sämtliche Felder werden jeweils mit einem leeren *SetOfBaskets* instantiiert. Für die Rückgabe der Ergebnisse werden noch die Instanzen instantiiert.
Danach startet der eigentliche Analyseprozess, der immer den gleichen Aufbau hat. Zunächst wird die Methode *generateCandidates* aufgerufen, welche zunächst alle potentiellen Artikelnummern, die eine Assoziationsregel repräsentieren könnten, speichert. Dazu werden die zu untersuchenden Instanzen von ihrem Start- bis zu ihrem Endindex durchlaufen und dem entsprechenden Datumsfeld im Array zugeordnet. Hierbei werden die einzelnen Warenkörbe identifiziert und sowohl die Anzahl der Artikel in einem einzelnen Warenkorb als auch die Gesamtanzahl der Warenkörbe an einem Tag in den entsprechenden Feldern in *Basket* und *SetOfBaskets* gespeichert. Bei einem Wechsel von einem elementaren Zeitintervall zum nächsten werden die Mindest-Support-Werte berechnet. Alle Einträge, die diesen nicht erfüllen, werden gelöscht. Dieses geschieht für alle Tage.
Bei jedem weiteren Durchlauf wird die Kandidatengenerierung nicht mehr auf den übergebenen Instanzen durchgeführt, sondern auf den Ergebnissen des vorangegangen Durchlaufes. Dieses führt zu einer starken Reduzierung der Ausführungszeit, ohne die Ergebnisse zu beeinträchtigen. Als nächstes wird *cleanUpStarPattern* aufgerufen, welches jedes Feld, das ein Wildcard-Muster repräsentiert, löscht. Dieses hat beim ersten Durchlauf keine Bedeutung, bei nachfolgenden Durchgängen stellt es jedoch sicher, dass keine alten Ergebnisse übernommen werden.

Im nächsten Schritt sollen alle Regeln gefunden werden, die möglicherweise für ein Wildcard-Muster halten. Dies geschieht mittels *updateStarPattern*. Hier werden für jedes elementare Zeitintervall aus dem Beobachtungszeitraum die gefundenen Kandidaten in die entsprechenden Wildcard-Muster mittels der *SetOfBaskets* Methode *fuzzyUpdate* kopiert. Dabei wird das Vorkommen einzelner Regeln an unterschiedlichen Tagen mittels *testStarpatternFuzzyMatch* mitgezählt und Regeln, die den vorgegebenen Fuzzy-Support nicht erfüllen durch die Methode testBasketsFuzzyMatch gelöscht.
Die Methode *updateResults* kopiert die gefundenen und gültigen Assoziationsregeln zusammen mit den erreichten Support-Werten in das Ergebnis-Array. Außerdem wird die Repräsentation von Wildcard-Mustern konvertiert. Danach wird der Zähler für die Anzahl der Artikel in einer Regel um den Wert Eins erhöht.
Der Algorithmus führt diese Befehlsabfolge solange durch, wie in dem Ergebnis-Array Daten eingetragen werden. Sobald keine Assoziationsregeln der geforderten Größe mehr gefunden werden, bricht der Algorithmus ab.
Die Ergebnisse können dann durch den Aufruf der Methode *getResults* in Form von Instanzen ausgelesen werden.

13.3.2 SetOfBaskets

Die Klasse SetOfBaskets soll die Regeln oder die Warenkörbe eines Tages repräsentieren. Dazu besteht sie aus einem *FastVector* von Baskets und einem Integer für die Anzahl der Warenkörbe an einem Tag. Der Wertebereich von Integer für die Anzahl der Warenkörbe sollte mit einer Maximalgröße von $2^{31} - 1$ ausreichend sein und der *FastVector* wurde aufgrund von Performanzgründen ausgewählt. Die Methoden wurden, wie in Abschnitt 11.2 beschrieben, umgesetzt.
Erweiterungen zu dem Entwurf gab es bezüglich des Fuzzy-Supports. Es wurde die Methode *fuzzyUpdate* eingeführt, die beim Einfügen von Warenkörben in ein SetOfBaskets an einem Wildcard-Muster den Zähler für die Anzahl der Vorkommen an diesem Tag mittels einer *Basket*-Methode inkrementiert.

13.3.3 Basket

Die Klasse Basket stellt einen einzelnen Warenkorb oder eine einzelne Regel dar. Sie beinhaltet dazu einen *FastVector* für die Artikelnum-

mern sowie ein Datum und einen Zähler. In Erweiterung zu dem Entwurf wurden Zähler für den Fuzzy-Support eingeführt. Der FuzzyCount und der Fuzzy-Support speichern zum einen das gesamte Aufkommen der repräsentierten Regel und zum anderen den Fuzzy-Support bei Wildcard-Mustern. Dazu existieren entsprechende set- und get- Methoden.

13.4 Fazit

Der Algorithmus stellt in seiner jetzigen Form dank Precise- und Fuzzy-Match und seiner ordentlichen Performanz bereits eine gute Einsatzvielfalt zur Verfügung.
Eine sinnvolle Erweiterung, die aufgrund der begrenzt verfügbaren Zeit unterblieb, wäre eine Anpassung der kalendarischen Muster auf eine feinere Zeitgranularität, wie zum Beispiel auf Stunden als elementare Zeitintervalle. Damit ließen sich mit Bezug auf die Personalisierung von Angeboten auch Verläufe im Kaufverhalten über einen Tag herausfinden.
Des Weiteren wäre eine Konfigurierbarkeit des Fuzzy-Supports für Wildcard-Muster denkbar, wenn anstatt den Fuzzy-Support in Abhängigkeit von der Anzahl der Tage zu betrachten, ihn von der Anzahl der Gesamtwarenkörbe, die an dem Wildcard-Muster galten, fest zu machen. Damit würden Tage mit wenigen Warenkörben nicht das gleiche Gewicht erhalten, wie Tage mit sehr vielen Warenkörben.
Insgesamt stellt der Algorithmus dank der verwendeten Datenstrukturen und der Umsetzung des Temporal-Apriori Ansatzes eine vernünftige Implementierung des Algorithmus zum Finden von kalendarischen Mustern dar und sollte für die Personalisierung brauchbare Ergebnisse liefern.

14 Technisches Handbuch: Framework

Dieses Handbuch beschreibt den Aufbau des Frameworks, seine Realisierung und seine Anwendungsmöglichkeiten. Das Framework dient dazu, die Data Mining Bibliothek WEKA [WF01] und ihre Erweiterungen durch die Projektgruppe mittels einer einfachen Konfigurationsdatei anzusteuern und so Analysen wie Clustering und das Finden von kalendarischen Mustern zu ermöglichen.
Das Handbuch richtet sich an Benutzer, die das Framework zum Data-Mining benutzen wollen, sowie an Entwickler, die das Framework um eigene Erweiterungen ergänzen wollen. Grundlegende Kenntnisse in den Bereichen KDD und Data Mining werden vorausgesetzt.
Der Rest dieses Handbuchs ist folgendermaßen aufgebaut: Zunächst werden in Abschnitt 14.1 die Anforderungen an das Framework genannt. Dann wird in Abschnitt 14.2 die grundlegende Architektur des Frameworks beschrieben. In Abschnitt 14.3 wird auf die konkrete Java-Implementierung des Frameworks eingegangen. Der darauf folgende Abschnitt gibt einen Überblick über alle Parameter, mit denen sich das Framework steuern lässt. Abschnitt 14.5 erläutert die Benutzung des Frameworks anhand zweier Anwendungsbeispiele. Der letzte Abschnitt bewertet das Framework bezüglich der Anforderungen und gibt einen Ausblick auf weitergehende Entwicklungsmöglichkeiten.

14.1 Anforderungen

Das hier beschriebene Framework stellt die Weiterentwicklung des in der Anforderungsdefinition der Analyse-Gruppe (Kapitel 10) beschriebenen Analyse-Systems dar. Dort wird erläutert, dass die Projektgruppe die Aufgaben der Analyse zunächst in einem einfachen System entwickeln will, in dem zunächst nur zwei Algorithmen mit fest vorgegebenen Daten ausgeführt werden.
Das Ziel der Projektgruppe ist jedoch die Entwicklung einer Bibliothek, die solche Analysen metadatengesteuert vornehmen kann. Die

Aufgaben des Frameworks sind demnach,

- das Auslesen der zu analysierenden Daten aus einer (möglichst beliebigen) Datenquelle,
- das Anwenden eines der in der Anforderungsdefinition der Analyse genannten Algorithmen (Clustering und temporale Assoziationsanalyse),
- sowie das Speichern der Analyseergebnisse in einer (ebenfalls möglichst beliebigen) Datensenke

durch Metadaten steuerbar zu machen.
Aufgrund der begrenzten Zeit, die der Projektgruppe für die Implementierung für mögliche Datenquellen, –senken und Algorithmen zur Verfügung steht, ist eine weitere wichtige Anforderung die leichte Erweiterbarkeit. So soll es möglich sein, den Funktionsumfang des Frameworks zu erweitern, ohne dafür das Framework selbst verändern zu müssen.

14.2 Architektur

Das Grundgerüst des Frameworks besteht aus vier Komponenten:

1. Zugriff auf Datenquellen:
   Der Zugriff auf Datenquellen wird über ein Iterator-Interface namens *ConfigurableSourceIterator* durchgeführt. Jede Implementation dieses Interfaces lässt sich über spezifische Parameter steuern.

2. Steuerung des anzuwendenden Algorithmus:
   Implementationen des Interfaces *AlgorithmController* regeln die korrekte Ansteuerung eines zu verwendenden Algorithmus.

3. Ausgabe in eine Datensenke:
   Die Ausgabe der erzeugten Analysedaten geschieht über das Interface *ResultWriter*.

4. Steuerung der obigen Komponenten:
   Das Zusammenspiel der einzelnen Komponenten wird durch die Klasse *AlgorithmRunner* gesteuert. Diese verarbeitet eine Reihe von Parametern und entscheidet dann, welche Datenquelle, welche Datensenke und welcher Algorithmus benutzt werden.

Mit diesen vier Komponenten lassen sich bereits eine Vielzahl von Analysen ausführen. Ein Beispiel dafür findet sich in Abschnitt 14.5.1, wo gezeigt wird, wie mit dem Framework ein einfaches Clustering durchgeführt werden kann. Abbildung 14.1 stellt den Daten- und

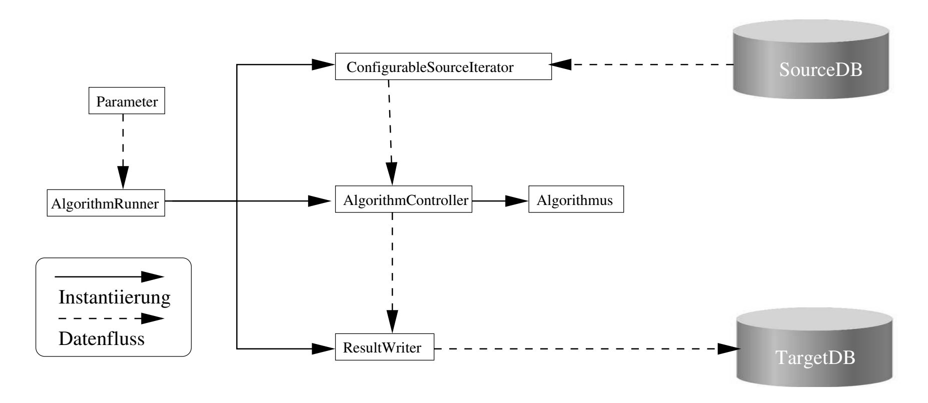


Abbildung 14.1: Daten- und Kontrollfluss im Framework

Kontrollfluss einer solchen Analyse dar.
In einigen Fällen kann es vorkommen, dass vor der eigentlichen Analyse bestimmte Vorbereitungsschritte (*Preprocessing*) und nach der Analyse bestimmte Nachbereitungsschritte (*Postprocessing*) erfolgen müssen. Weiterhin kann es in einigen Fällen gewünscht sein, mehrere solcher komplexer Analysen automatisch hintereinander auszuführen. Auch dies ist mit dem Framework leicht realisierbar. Hierfür dienen die folgenden Komponenten:

1. Vor- und Nachbereitungen lassen sich über das Interface *PrePostProcessor* definieren. Dieses definiert einzig die Methode *run()*, in der ein *PrePostprocessor* beliebige Berechnungen durchführen kann.

2. Zusammen mit einem *AlgorithmRunner* lassen sich beliebig viele Pre- und Postprozessoren zu Analysen zusammenfassen. Wird solch eine Analyse gestartet, werden zunächst der Reihe nach die Preprozessoren, dann der *AlgorithmRunner* und schließlich die Postprozessoren ausgeführt.

3. Analysen können über die Klasse *AnalysisRunner* zusammengefasst und nacheinander ausgeführt werden.

Eine schematische Darstellung der Ausführung solcher komplexer Analysen findet sich in Abbildung 14.2.

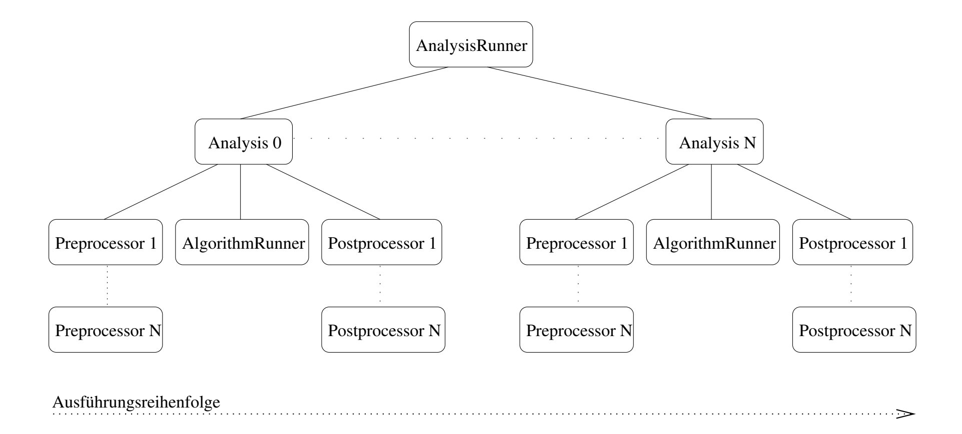


Abbildung 14.2: Ausführung komplexer Analysen

14.2.1 Steuerung des Frameworks

Das Framework kann komplett über Parameter gesteuert werden. Wird das Framework direkt aus einem eigenen Java-Programm verwendet, können die Parameter in Form von *java.util.Properties* übergeben werden. Soll das Framework stattdessen als eigenständige Applikation verwendet werden, werden die Parameter in einer Konfigurationsdatei angegeben. Das Format dieser Datei ist durch die *java.util.Properties* Methode *load* [Mic03] vorgegeben. Eine genaue Beschreibung der jeweils benötigten Parameter findet sich in Abschnitt 14.4.

14.2.2 Auslesen der Daten aus der Datenquelle

Um die Daten aus der Datenquelle zu lesen, instantiiert der *AlgorithmRunner* einen passenden *ConfigurableSourceIterator*. Dieser wird ebenso wie der *AlgorithmRunner* über Parameter der gleichen Art konfiguriert. Das Auslesen aus der Datenquelle erfolgt dann inkrementell über die Methoden *hasNext()* und *next()*.

14.2.3 Steuerung des Algorithmus

Basierend auf den Parametern entscheidet der *AlgorithmRunner*, welche Implementation des Interface *AlgorithmController* instantiiert und verwendet wird. Der jeweilige *AlgorithmController* bekommt dann vom *AlgorithmRunner* die Quelldaten übergeben und startet den anzuwendenden Algorithmus.

14.2.4 Speichern der Analysedaten

Zum Speichern der Analysedaten instantiiert der *AlgorithmRunner* eine Implementation des Interface *ResultWriter* und übergibt dieser die vom Algorithmus erzeugten Analysedaten. Der *ResultWriter* speichert diese dann in der jeweiligen Datensenke. Auch hier werden die Parameter durch den *AlgorithmRunner* weitergereicht.

14.3 Implementierung

Das Framework wurde in Java mit dem Sun SDK in der Version 1.4 entwickelt. Um die leichte Erweiterbarkeit zu garantieren, wurden zunächst eine Reihe von Interfaces erstellt, die das Grundgerüst des Frameworks darstellen. Diese werden in Abschnitt 14.3.1 beschrieben. Für eine sinnvolle Fehlerbehandlung wurden spezielle Exceptions erstellt, die in Abschnitt 14.3.2 beschrieben werden. Für die Interfaces stehen eine Reihe von Implementierungen zur Verfügung, die die notwendige Funktionalität für Standard-Datenquellen und -senken, wie z.B. über JDBC erreichbare Datenbanken bereitstellen. Die vorhandenen Implementierungen werden in Abschnitt 14.3.3 beschrieben. Eine Anleitung, wie selbst entwickelte Implementierungen mit dem Framework benutzt werden können, um z.B. eine Anbindung an XML-Datenbanken zu erstellen, findet sich in Abschnitt 14.3.4.

14.3.1 Interfaces

Das Grundgerüst des Frameworks basiert auf einer Reihe von Interfaces, die die Aufgaben der einzelnen Framework-Bausteine definieren. Durch die konsequente Verwendung dieser Interfaces wird die leichte Erweiterbarkeit des Frameworks sichergestellt, da zur Laufzeit konfigurierbar ist, welche konkreten Klassen verwendet werden. Das Framework enthält die folgenden Interfaces:

OptionHandler: Das Interface *OptionHandler* ist das Grundgerüst für das Verarbeiten von Optionen. Als solches wird es von allen im Folgenden beschriebenen Interfaces erweitert.

ConfigurableSourceIterator: Der *ConfigurableSourceIterator* ist das Interface, das zur Einbindung der Datenquellen verwendet wird. Es erbt sowohl vom *OptionHandler* als auch von *SourceIterator* (ein Interface aus der von der Projektgruppe modifizierten Version von WEKA, siehe Kapitel 11).

MetaDataProvider: Das Interface *MetaDataProvider* kann ein *ConfigurableSourceIterator* implementieren, sofern er in der Lage ist, Auskunft über die erzeugten Attribute zu geben. Falls eine Datenquelle dieses Interface implementiert, übernimmt der *AlgorithmRunner* die Metadaten von ihr. Ein Beispiel dafür ist der *ArffFileIterator* (siehe Seite 114).

AlgorithmController: Das Interface *AlgorithmController* dient dazu, einen bestimmten Analyse-Algorithmus auszuführen. Jeder Algorithmus benötigt dabei eine eigene Implementierung[1], die jeweils unterschiedliche Parameter verarbeiten kann.

ResultWriter: Das *ResultWriter* Interface wird benutzt, um die erzeugten Analysedaten in eine Datensenke zu speichern.

PrePostProcessor: Um in komplexen Analysen beliebige Vor- und Nachbereitungsschritte ausführen zu können, bedient sich das Framework dieses Interfaces.

IAlgorithmRunner: Das Interface *IAlgorithmRunner* spezifiziert die Schnittstelle zu einfachen Analysen, die ohne Pre- und Postprozessoren auskommen.

IAnalysisRunner: Dieses Interface spezifiziert die Schnittstelle zu komplexen Analysen.

Detaillierte Beschreibungen der einzelnen Methoden der Interfaces finden sich in den Javadoc Kommentaren der Implementierung. Diese sind unter [Dik03] verfügbar. Abbildung 14.3 stellt die Interfaces

[1] die sich teilweise aber sehr ähnlich sind; beispielsweise gibt es für die verschiedenen Clusterer aus WEKA eine abstrakte Klasse, die den Großteil der Funktionalität bereitstellt und für jeden Clusterer jeweils eine davon erbende Klasse, die sich lediglich um die Instantiierung und die Verarbeitung von Parametern kümmert.

und deren Beziehung untereinander dar. Zusätzlich sind die beiden Klassen *AnalysisRunner* und *AlgorithmRunner* dargestellt, die über *main*-Methoden verfügen und damit von Benutzern direkt ausführbar sind.

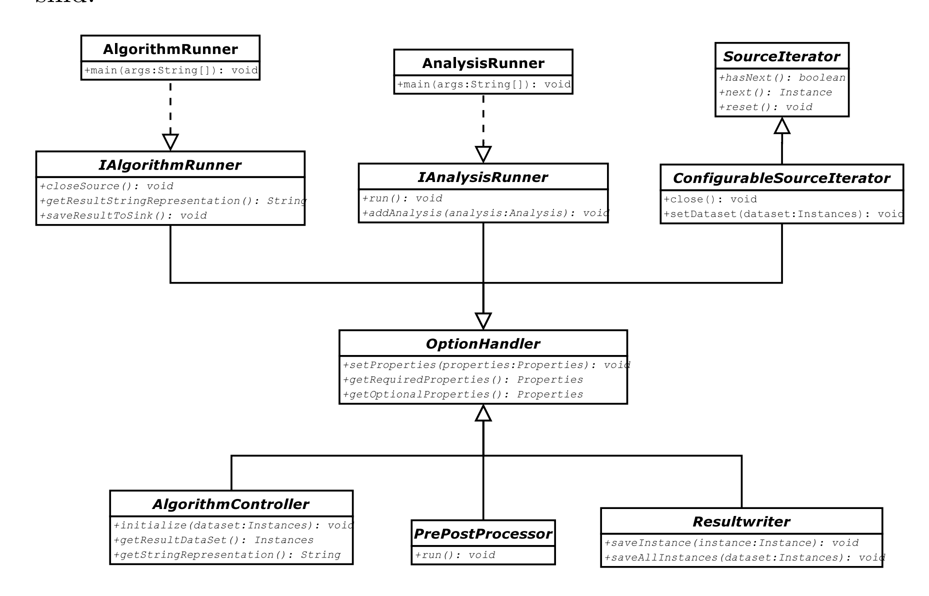


Abbildung 14.3: Klassendiagramm

14.3.2 Exceptions

Um verschiedene Fehlersituationen sauber behandeln zu können, sind im Framework einige Exceptions definiert worden. Dies sind im Einzelnen:

PropertyException: Die *PropertyException* ist die Basisklasse für alle Exceptions, die auftreten können, wenn Parameter fehlen oder mit falschen Werten belegt sind.

MissingPropertyException: Eine Unterklasse der *PropertyException*. Wird immer dann geworfen, wenn zwingend notwendige Parameter nicht angegeben wurden.

IllegalPropertyException: Ebenfalls eine Unterklasse der *PropertyException*. Wird immer dann geworfen, wenn ein angegebener Parameter einen illegalen Wert enthält.

SourceNotAvailableException: Diese Exception wird von einem *SourceIterator* geworfen, wenn er nicht in der Lage ist, auf seine Datenquelle zuzugreifen, z.B. weil eine Datenbankverbindung nicht aufgebaut oder eine Datei nicht geöffnet werden kann.

SinkNotAvailableException: Wird von einem *ResultWriter* geworfen, wenn er die Ergebnisse nicht in die Datensenken schreiben kann.

14.3.3 Vorhandene Implementierungen

In Abschnitt 14.3.1 wurden die Interfaces beschrieben, die das Grundgerüst des Framework darstellen. Dieser Abschnitt beschreibt die vorhandenen Implementierungen dieser Interfaces. Zunächst wird auf die unterstützten Datenquellen eingegangen, danach werden die implementierten Datensenken und abschließend die vorhandenen Algorithmen beschrieben.

Datenquellen

Im Package *diko.framework.input* befinden sich die vorhandenen Implementierungen für den Zugriff auf Datenquellen. Zur Zeit sind dies zwei: eine ermöglicht den Zugriff auf ARFF-Dateien, die andere den Zugriff auf relationale Datenbanken, die über JDBC ansprechbar sind.

Zugriff auf ARFF-Dateien Die Klasse *ArffFileIterator* ermöglicht den Zugriff auf ARFF-Dateien. ARFF-Dateien sind die Standard-Datenquelle von WEKA. Da ARFF-Dateien neben den eigentlichen Daten auch Metadaten über die Struktur der Daten enthalten, implementiert der *ArffFileIterator* zusätzlich zum Interface *ConfigurableSourceIterator* auch noch das Interface *MetaDataProvider*.
Der *ArffFileIterator* benötigt als einzige Option den Dateinamen der einzulesenden ARFF-Datei. Falls er die angegebene Datei nicht lesen kann, wird eine *SourceNotAvailableException* (siehe Abschnitt 14.3.2) geworfen.

Zugriff auf JDBC Datenbanken Der *ConfigurableJDBCSourceIterator* erlaubt Zugriff auf Datenbanken, die über die JDBC-Schnittstelle ansprechbar sind. Sie benötigt Optionen, die die Verbindungsparameter (Datenbanktreiber, Login usw.) sowie ein gültiges SQL-Statement

spezifizieren. Optional ist es möglich, anzugeben, welche der ausgelesenen Spalten als temporale Daten erkannt werden. In diesem Fall erzeugt der *ConfigurableJDBCSourceIterator* temporale Instanzen anstelle der standardmäßig verwendeten normalen Instanzen.

Datensenken

Die verfügbaren Datensenken befinden sich im Package *diko.framework.output*. Es gibt Implementierungen für ARFF-Dateien, JDBC-Datenbanken und eine einfache Implementierung, die die Daten auf die Standardausgabe schreibt.

Speichern in ARFF-Dateien Die Klasse *ArffWriter* ermöglicht es, die erzeugten Analysedaten in einer herkömmlichen ARFF-Datei zu speichern. Auf diese Weise ist es z.B. leicht möglich, die vom Framework erzeugten Ergebnisse mit der Original-Version von WEKA weiter zu verarbeiten.
Die Klasse arbeitet auch ohne Optionen, schreibt die erzeugten ARFF-Daten dann aber nicht in eine Datei, sondern auf die Standardausgabe. Wird ein Dateiname als Option angegeben, werden die Daten in diese Datei gespeichert.

Speichern in JDBC-Datenbanken Sollen die Ergebnisse in einer JDBC Datenbank gespeichert werden, kann die Klasse *JDBCWriter* dafür verwendet werden. Diese speichert die erzeugten Daten in der per Option angegebenen Tabelle. Es muss dabei sichergestellt sein, dass die Daten zur Tabelle passen, d.h. die Anzahl der Spalten muss identisch sein.
Der *JDBCWriter* ermöglicht auch das Speichern von Instanzen, die Vektor-Attribute (siehe Abschnitt 14.3.5) enthalten. Da er aber nur das Schreiben in eine einzelne Tabelle unterstützt, bedient er sich eines speziellen Verfahrens, um die multidimensionalen Daten in einer Tabelle speichern zu können. Dabei wird für jeden Wert eines Vektor-Attributs eine eigene Zeile in der Tabelle angelegt. Das folgende Beispiel verdeutlicht dies:

Beispiel Angenommen, die erzeugten Daten hätten drei Attribute, davon das erste numerisch, die anderen zwei vom Typ Vektor. Soll nun eine Instanz gespeichert werden, die die Werte $< 1, \{2, 3\}, \{4, 5, 6\} >$ hat, wird folgende Tabelle erzeugt:

Spalte 1	Spalte 2	Spalte 3
1	2	4
1	3	5
1	NULL	6

Für diese Instanz werden also drei Zeilen in der Tabelle erzeugt, da der größte Vektor der Instanz drei Elemente enthält.

Auf diese Weise kann auch bei vielen Vektor-Attributen eine platzsparende Speicherung erreichet werden. Die Alternative, alle Werte der Vektor-Attribute zu permutieren und für jede Permutation eine Zeile in der Tabelle anzulegen, würde im vorliegenden Beispiel schon sechs Zeilen in Anspruch nehmen. Diese Art der Speicherung erfordert es, dass die Spalten, in denen die Vektor-Attribute gespeichert werden sollen, `NULL` Werte erlauben.

Standardausgabe Sollen die erzeugten Daten lediglich schnell angesehen werden, oder vielleicht per Pipeline einer anderen Anwendung zur Verfügung gestellt werden, kann der *SimpleStdoutWriter* verwendet werden. Dieser schreibt jede zu speichernde Instanz einfach per *System.out.println()* in die Standardausgabe. Daher benötigt er auch keine Parameter.

Algorithmen

Dieser Abschnitt beschreibt, welche Algorithmen mit dem Framework gesteuert werden können. Generell soll es möglich sein, mit dem Framework beliebige Algorithmen zu steuern. In der Praxis beschränkt sich dies momentan auf Algorithmen, die nur eine Datenquelle als Eingabe benötigen. Dies ist dadurch begründet, dass es im Rahmen der Projektgruppe nicht erforderlich ist, Algorithmen zu verwenden, die mehrere Datenquellen benötigen. Ein Beispiel hierfür sind Klassifikatoren, die sowohl Trainingsdaten als auch die eigentlich zu untersuchenden Daten benötigen.
Im Folgenden werden die im Framework benutzbaren Algorithmen vorgestellt.

Clustering Das Framework bietet standardmäßig vier in WEKA implementierte Clusterer an. Dabei handelt es sich um SimpleK-Means [WF01], XMeans [PM00], Erwartungsmaximierung [WF01]

sowie Cobweb [Fis87]. Alle diese Algorithmen verfügen über eine Reihe von optionalen Parametern, wie z.B. die Anzahl der zu findenden Cluster usw.
Weiterhin ist es möglich, bestimmte Attribute der Eingabedaten vor den Clusterern zu „verstecken“, so dass diese nicht berücksichtigt werden. In der Ergebnisdarstellung werden diese Attribute jedoch wieder berücksichtigt. Auf diese Weise ist es z.B. möglich, Datensätze anhand einer ID zu identifizieren, ohne dass diese das Analyseergebnis beeinflusst.
Alle Clusterer produzieren Ergebnisse der gleichen Struktur. Diese ergibt sich aus der Struktur der Eingabedaten erweitert um ein numerisches Attribut, welches die Cluster-ID eines Datensatzes darstellt.

Temporale Assoziationsanalyse Neben den Clusteralgorithmen bietet das Framework einen Algorithmus zum Finden von temporalen Assoziationsregeln, der von der Projektgruppe selbst implementiert wurde (siehe Kapitel 13). Dieser benötigt als Eingabedaten eine Relation von temporalen Instanzen, die zwei Attribute besitzen. Das erste Attribut spezifiziert die Kundennummer des Käufers, das zweite Attribut die Nummer des gekauften Produktes. Die Transaktionszeit der temporalen Instanz stellt den Kaufzeitpunkt dar.
Dieser Algorithmus erzeugt eine Ergebnisrelation mit sechs Attributen. Die ersten drei Attribute stellen die Kalendermuster dar, für die eine Regel gilt. Das nächste Attribut bestimmt den Support einer Regel. Das vorletzte Attribut ist eine fortlaufende Nummer, mit der die Regel identifiziert werden kann. Das letzte Attribut ist vom Typ Vektor und enthält die in der Regel enthaltenen Produkte.

Beispiel Angenommen, der Algorithmus entdeckt in den Eingabedaten als erstes eine Regel, die besagt, dass an jedem Tag im August des Jahres 2003 die Produkte mit der ID 1 und 2 zusammen gekauft werden. Sei weiterhin der Support dieser Regel 0,75, dann enthält die Ergebnisrelation die folgende Instanz:

$$< 2003, 8, -1, 0.75, 1, \{1, 2\} >$$

Dabei steht -1 für ein sogenanntes *Star-Pattern*, also in diesem Fall für jeden Tag.

Kopieren der Eingabedaten Das Framework bietet neben den eigentlichen Analysealgorithmen auch eine Möglichkeit, die Eingabe-

daten unverändert an die Datensenke weiterzuleiten. Das ist zum Beispiel von Nutzen, wenn Daten aus einer relationalen Datenbank in eine ARFF-Datei überführt werden sollen. Hierzu dient der *SimpleTestController*.

14.3.4 Eigene Implementierungen einbinden

Ein wesentlicher Aspekt des Frameworks ist die leichte Erweiterbarkeit mit eigenen Datenquellen und -senken, Algorithmen, Pre- und Postprozessoren. Diese leichte Erweiterbarkeit wird durch die Fähigkeit von Java erreicht, Klassen dynamisch anhand ihres Namens zu instantiieren. Dadurch ist es möglich, den Kontrollfluss des Frameworks komplett über die Verwendung der Interfaces zu steuern, ohne (im Code des Frameworks) jemals zu wissen, welche konkreten Klassen die Arbeit der einzelnen Schritte übernehmen. Dies wird erst zur Laufzeit dynamisch aufgrund der jeweiligen Parameter entschieden.
Aufgrund dessen ist es für Anwender des Frameworks sehr einfach, dieses mit eigenen Implementierungen zu erweitern, ohne dass dazu am Framework selbst etwas geändert werden muss. Soll beispielsweise eine neue Datenquelle an das Framework angebunden werden, muss eine Klasse, die das Interface *ConfigurableSourceIterator* implementiert und über einen gültigen Default-Konstruktor verfügt, geschrieben werden. Um die Klasse in einen Lauf des Frameworks einzubinden, ist es lediglich notwendig, ihren vollständigen Namen als Parameter *sink* dem *AlgorithmRunner* zu übergeben (siehe Seite 119).

14.3.5 Notwendige Änderungen an WEKA

Um alle Funktionen des Frameworks implementieren zu können, mussten einige Änderungen an WEKA vorgenommen werden. Neben dem Umbau der Klasse *Instances*, die nun nicht mehr alle Daten im Hauptspeicher vorhält (siehe Kapitel 12), ist vor allem die Erweiterung der Attribute um den Typ Vektor notwendig gewesen, um die temporalen Assoziationsregeln sinnvoll speichern zu können.
Der Grund dafür ist, dass in den Assoziationsregeln eine beliebige Anzahl an Produkten vorkommen kann. WEKA kann in der ursprünglichen Version jedoch nur Instanzen mit einer festen Anzahl an Attributen verwalten. Daher hat die Projektgruppe einen neuen Typ von Attributen eingeführt – die Vektor-Attribute.
Vektor-Attribute ermöglichen es, in einem einzelnen Attribut mehrere

Werte zu speichern. So ist es z.B. möglich, ein Attribut zu definieren, welches eine Reihe von Produkt-IDs aufnimmt. Auf diese Weise sind auch Assoziationsregeln mit mehreren Produkten in WEKA möglich. Eine genaue Beschreibung, wie die Vektor-Attribute implementiert wurden, lässt sich den Javadoc-Kommentaren in [Dik03] entnehmen.

14.4 Parameter der vorhandenen Implementierungen

Dieser Abschnitt listet alle im Framework verfügbaren Parameter auf. Zunächst werden die notwendigen Parameter des *AlgorithmRunner* beschrieben, der zur Durchführung einfacher Analysen benutzt wird. Im Anschluss werden die Parameter der dabei verwendbaren vorhandenen Datenquellen, -senken und Analysealgorithmen beschrieben. Zum Abschluss wird erläutert, welche Parameter benötigt werden, um mit Hilfe des *AnalysisRunner* komplexe Analysen durchzuführen.

14.4.1 AlgorithmRunner

Jede einfache Analyse wird durch die Klasse *AlgorithmRunner* gesteuert. Um dies auch außerhalb des eigenen Java-Codes zu ermöglichen, verfügt der *AlgorithmRunner* über eine *main* Methode. Diese erwartet als Parameter den Namen einer Konfigurationsdatei. In ihr werden alle für die Durchführung der Analyse benötigten Parameter definiert. Dazu gehören sowohl die Parameter, die der *AlgorithmRunner* selbst auswertet, als auch die, die er an die beteiligten Datenquellen, -senken und Algorithmen weitergibt.
Der *AlgorithmRunner* selbst benötigt folgende Parameter:

`source`: Bestimmt, welche Klasse die Anbindung an die Datenquelle realisiert. Es muss der volle Klassenname angegeben werden, wie z.B. *diko.weka.input.ArffFileIterator*. Die angegebene Klasse muss das Interface *ConfigurableSourceIterator* implementieren.

`sink`: Bestimmt, welche Klasse die Anbindung an die Datensenke übernimmt. Es muss ebenfalls der volle Klassenname angegeben werden. Die Klasse muss das Interface *ResultWriter* implementieren.

`algorithm`: Voller Klassenname eines *AlgorithmController*.

Um Informationen über die in den Quelldaten enthaltenen Attribute zu erhalten, bietet der *AlgorithmRunner* zwei Möglichkeiten: Entweder stellen die Datenquellen diese Informationen selbst bereit, oder sie müssen vom Benutzer in Form von Parametern angegeben werden. Wird eine Datenquelle verwendet, die in der Lage ist, diese Informationen bereitzustellen, muss der dafür zuständige *ConfigurableSourceIterator* zusätzlich das Interface *MetaDataProvider* implementieren. Ist dies der Fall, benutzt der *AlgorithmController* die Methoden des *MetaDataProvider*, um Daten über die enthaltenen Attribute und den Namen der Relation zu erfahren.
Für den Fall, dass die benutzte Datenquelle keine Informationen über die Attribute liefern kann, müssen diese Daten zusätzlich in der Konfigurationsdatei angegeben werden. Dazu werden für jedes Attribut zwei Parameter benötigt:

`attributeName_X`: Definiert den Namen des Attributs.

`attributeType_X`: Definiert den Typ des Attributs.

Dabei ist `X` der Index des Attributs, das erste Attribut hat den Index 0, das zweite 1 usw.

14.4.2 Datenquellen

Die im Framework implementierten Datenquellen benötigen ebenfalls eine Reihe von Optionen, um korrekt arbeiten zu können. Diese sind im Folgenden aufgeführt.

ArffFileIterator

Der *ArffFileIterator* benötigt als einzigen Parameter den Namen der zu öffnenden Datei.

`arffFile`: Der Name der zu ladenden Datei, inklusive Pfad.

ConfigurableJDBCSourceIterator

Die benötigten Optionen für den Zugriff auf Datenbankmanagementsysteme via JDBC sind:

`source.jdbcDriver`: Klassenname des JDBC-Treibers, der den Zugriff auf die Datenbank bereitstellt.

`source.jdbcConnection`: Der URL, den der Treiber benötigt, um die Datenbank-Verbindung auszubauen.

`source.jdbcLogin`: Das Login für die Datenbank.

`source.jdbcPassword`: Das Passwort.

`source.jdbcSQL`: Das SQL-Statement, das die einzulesenden Daten liefert.

Die folgenden Parameter sind optional:

`source.tza`: Falls dieser Parameter gesetzt ist, bestimmt er die Spalte der Eingabedaten, die den Transaktionszeitanfang repräsentiert. Der Wert dieses Parameters muss ein Integer im Bereich 0 bis (Anzahl der Spalten - 1) sein.

`source.tze`: Bestimmt analog die Spalte, die das Transaktionszeitende repräsentiert.

`source.gza`: Bestimmt die Spalte, die den Gültigkeitszeitanfang repräsentiert.

`source.gze`: Bestimmt die Spalte, die das Gültigkeitszeitende repräsentiert.

Falls einer der obigen optionalen Parameter gesetzt ist, erzeugt der *ConfigurableJDBCSourceIterator* an Stelle der standardmäßig erzeugten *Instance* eine *TemporalInstance* beim Aufruf von *next()*.

14.4.3 Datensenken

Nachfolgend sind die Parameter für die im Framework enthaltenen Datensenken aufgeführt.

ArffWriter

Der *ArffWriter*, der zum Erzeugen von ARFF-Dateien verwendet wird, hat keine notwendigen Parameter. Der einzige optionale Parameter ist:

`arffWriter.File`: Ist dieser Parameter gesetzt, bestimmt er den Dateinamen, in dem die ARFF-Datei gespeichert wird. Wird dieser Parameter nicht gesetzt, gibt der *ArffWriter* die erzeugten Daten auf der Standardausgabe aus.

JDBCWriter

Der *JDBCWriter* benötigt die folgenden Parameter:

`sink.jdbc{Driver,Connection,Login,Password}`: analog zu den jeweiligen `source.*` Parametern des *ConfigurableJDBCSourceIterator* (Seite 120f).

`sink.jdbcTable`: Der Name der Tabelle, in die die Ergebnisse gespeichert werden sollen.

Weiterhin gibt es den folgenden optionalen Parameter:

`sink.jdbcIgnoreAttributeNames`: Ist dieser Parameter mit einem (beliebigen) Wert belegt, entscheidet der *JDBCWriter* nicht anhand der Namen der Attribute, welches Attribut in welche Spalte der Datenbanktabelle geschrieben wird, sondern anhand der Position der Attribute. Dies ist notwendig, wenn die Spaltennamen der Tabelle nicht mit den Namen der im Framework verwendeten Attribute übereinstimmen.

SimpleStdoutWriter

Der *SimpleStdoutWriter* benötigt keine Parameter; er schreibt die Ergebnisse einfach in die Standardausgabe.

14.4.4 Algorithmen

Das Framework kann eine Reihe von Algorithmen auf die Eingabedaten anwenden, um Analysen durchzuführen. Welcher Algorithmus in einer Analyse durchgeführt wird, wird durch Kontrollklassen definiert, die das Interface *AlgorithmController* implementieren. Für jeden vom Framework ausführbaren Algorithmus gibt es eine entsprechende Kontrollklasse. Diese sorgen dafür, dass der jeweilige Algorithmus korrekt instantiiert und konfiguriert wird.

Clusterer

Das Framework kann eine Reihe Clusterer von WEKA ansteuern. Diese sind SimpleKMeans, XMeans, Erwartungsmaximierung und Cobweb. Die Kontrollklassen für die Clusterer sind so aufgebaut, dass

sie die Clusterer zunächst in ihrer Default-Konfiguration initialisieren. Um optimale Ergebnisse zu erhalten ist es jedoch oft notwendig, den Algorithmen bestimmte Parameter, z.B. die Anzahl der zu findenden Cluster vorzugeben. Dazu verarbeiten die einzelnen Kontrollklassen eine Reihe von optionalen Parametern, die im Folgenden aufgeführt sind.
Alle Clusterer beachten den gemeinsamen Parameter:

`clusterer.FilterColumns`: Mit diesem Parameter lässt sich spezifizieren, welche Spalten der Eingabedaten der Clusterer ignorieren soll. Sollen beispielsweise die einzelnen Datensätze mittels eines eindeutigen Schlüssels („ID") gekennzeichnet werden, ist es üblicherweise nicht sinnvoll, dass dieser Einfluss auf das Ergebnis des Clusterers hat. Die ID soll aber in den Ergebnisdaten enthalten sein, um die einzelnen Datensätze nach dem Clustering wieder identifizieren zu können. Sind beispielsweise Eingabedaten mit zwei Spalten zu clustern, von denen die erste eine ID, die zweite die eigentlich zu analysierenden Daten enthält, ist dieser Parameter mit dem Wert `1` zu spezifizieren. Um mehrere Spalten angeben zu können, sind diese mit den Zeichen „," und „-" voneinander zu trennen. Dabei bedeutet `clusterer.FilterColumns=1,2` z.B., dass die Spalten 1 und 2 ignoriert werden, `clusterer.FilterColumns=1-3` bedeutet, dass die Spalten 1, 2 und 3 ignoriert werden.

Im Folgenden sind die für die verschiedenen Clusterer spezifischen Parameter aufgeführt.

SimpleKMeans Der SimpleKMeans Clusterer unterstützt die folgenden Parameter:

`kMeans.numClusters`: Bestimmt, wie viele Cluster der Algorithmus finden soll.

`kMeans.seed`: Die zur Initialisierung des Zufallszahlengenerators verwendete Zahl.

XMeans Der XMeans Clusterer unterstützt die folgenden Parameter:

`XMeans.distanceFunction`: Klassenname der zu benutzenden Distanzfunktion. Default ist *weka.core.EuclideanDistance*.

`XMeans.distanceValue`: Wert, der benutzt wird, um die Distanz zwischen zwei unterschiedlichen nominalen Attributen zu bestimmen.

`XMeans.maxIterations`: Maximale Anzahl von Iterationen. Default ist 1.

`XMeans.maxKMeans`: Maximale Anzahl von Iterationen im „Improve-Parameter"-Teil der KMeans-Schleife. Default ist 1000.

`XMeans.maxKMeansStructure`: Maximale Anzahl von Iterationen im „Improve-Structure"-Teil der KMeans-Schleife. Default ist 1000.

`XMeans.maxNumClusters`: Die maximale Anzahl zu findender Cluster.

`XMeans.minNumClusters`: Die minimale Anzahl zu findender Cluster.

`XMeans.cutOffFactor`: Gibt an, welche Prozentzahl der besten Splits verwendet wird, wenn keine der neuen Centroide besser sind.

Erwartungsmaximierung Der Algorithmus EM unterstützt den folgenden Parameter:

`em.numClusters`: Bestimmt die Anzahl der zu findenden Cluster.

Cobweb Der Clusterer Cobweb unterstützt die folgenden Parameter:

`cobweb.Acuity`: Die minimale Standardabweichung.

`cobweb.Cutoff`: Der minimale Nutzwert.

Temporale Assoziationsregeln

Der Algorithmus zum Finden von temporalen Assoziationsregeln benötigt die folgenden Parameter:

`tempAssociations.minDate`: Das Datum, von dem an in den Eingabedaten nach Regeln gesucht wird. Muss in der Form `tt.mm.jjjj` angegeben werden.

`tempAssociations.maxDate`: Das Datum, bis zu dem nach Regeln gesucht werden.

`tempAssociations.minSupport`: Der Mindestsupport, den gefundene Regeln mindestens erfüllen müssen. Es muss gelten: $0 < minSupport \leq 1$.

Weiterhin beachtet der Algorithmus die folgenden optionalen Parameter:

`tempAssociations.fuzzySupport`: Wenn dieser Parameter angegeben wird, bestimmt der den Prozentsatz von Zeiteinheiten, an denen eine Regel gelten muss, damit sie als *Star-Pattern* gilt. Es muss gelten: $0 < fuzzySupport \leq 1$. Der Default ist 1.

`tempAssociations.initialRuleId`: Wenn dieser Parameter angegeben wird, wird die erste erzeugte Regel mit dem angegeben Wert als ID belegt. Nachfolgende Regeln erhalten jeweils eine um 1 inkrementierte ID. Der angegebene Wert muss eine Zahl im Wertebereich des Java Datentyps *int* sein. Dieser Parameter ist nützlich, wenn die erzeugten Regeln in einer Datenbanktabelle, welche bereits Regeln aus einem vorangegangenen Lauf des Frameworks enthält, gespeichert werden sollen.

`tempAssociations.debug`: Wenn dieser Parameter angegeben wird, gibt der Algorithmus eine Reihe von Debugmeldungen auf der Standardausgabe aus.

14.4.5 AnalysisRunner

Wie in Abschnitt 14.2 beschrieben, kann das Framework dazu benutzt werden, einfache Analysen zu komplexen zusammenzufassen. Dazu wird die Klasse *AnalysisRunner* verwendet. Sie ermöglicht es, n Analysen hintereinander auszuführen, wobei eine Analyse immer aus m Pre- oder Postprozessoren und einer einfachen Analyse (gesteuert durch den *AlgorithmRunner*) besteht. Dazu werden die folgenden Parameter benötigt:

`analysis_`i`.properties`: Dateiname einer Property-Datei, in der die für die i-te Analyse benötigten Parameter spezifiziert sind.

`analysis_`i`.preprocessor_`j: Vollständiger Klassenname des j-ten *Preprocessor* der i-ten Analyse.

`analysis_i.postprocessor_j`: Vollständiger Klassenname des j-ten *Postprocessor* der i-ten Analyse.

Die Indizes für die Analysen müssen jeweils bei 0 beginnen und „dicht“ sein, d.h. ohne die Angabe einer Analyse mit dem Index 1 wird eine angegebene Analyse mit einem Index $i > 1$ nicht beachtet. Gleiches gilt für die Pre- und Postprozessoren.

14.5 Anwendungsbeispiele

In diesem Abschnitt wird exemplarisch die Durchführung zweier Analysen beschrieben. Im folgenden Abschnitt wird zunächst eine einfache Analyse dargestellt, in Abschnitt 14.5.2 eine komplexe Analyse mit Pre- und Postprozessoren.

14.5.1 Einfache Analyse mittels AlgorithmRunner

Dieser Abschnitt beschreibt am Beispiel des Anwendungsfalles von Clustering mittels SimpleKMeans auf Daten aus einer JDBC-Datenbank die Funktionsweise des Frameworks bei der Durchführung einer einfachen Analyse. Im beschriebenen Fall werden die Daten aus einer JDBC-Datenbank geladen, dem Cluster-Algorithmus SimpleKMeans übergeben und die vom Algorithmus produzierten Ergebnisse im ARFF-Format auf die Standardausgabe ausgegeben.
Um dem konfigurierbaren Framework mitzuteilen, welche Aufgaben durchzuführen sind, sind diese in Form einer Property-Datei (wie in Abschnitt 14.4 beschrieben) in Schlüsseln und Werten zu formulieren. Eine beispielhafte Property-Datei ist in Abbildung 14.4 dargestellt.
Zunächst wird der Iterator spezifiziert, der für das Einlesen der Daten benötigt wird. Da im Beispiel aus einer JDBC-Datenquelle gelesen wird, ist es in diesem Fall ein *ConfigurableJDBCSourceIterator*. Hierfür dient der Schlüssel *source*.
Danach erfolgt die Benennung der Relation mit der Zeichenkette „Händler“ über den Schlüssel *relation*. Diese Zeichenkette wird später von WEKA intern für die Benennung der *Instances*-Klasse genutzt.
Anschließend wird der Name des zu ladenden Java-JDBC-Treibers für Oracle angegeben. Dieser wird für den Aufbau einer Verbindung zu Oracle aus Java heraus benötigt. Hierfür ist der Schlüssel mit der Bezeichnung *source.jdbcDriver* zuständig. Ebenso werden der Connection-String (Schlüssel *source.jdbcConnection*), der Benutzername (Schlüssel

```
#ein Kommentar beginnt mit #
source= diko.framework.input.ConfigurableJDBC\
  SourceIterator
relation= Händler
source.jdbcDriver= oracle.jdbc.driver.\
  OracleDriver
source.jdbcConnection= jdbc:oracle:thin:@\
  power2.offis.uni-oldenburg.de:1521:power2
source.jdbcLogin= cardprovider
source.jdbcPassword= *****************
#Lange Zeilen können mit \ weitergeführt werden
#Wichtig: das Leerzeichen am Ende.
#Leerzeichen am Anfang werden ignoriert.
source.jdbcSQL= select haendlerid, name \
  from haendler
attributeName_0= haendlerid
attributeType_0= numeric
attributeName_1= name
attributeType_1= nominal
algorithm=diko.framework.algorithms.\
  SimpleKMeansController
clusterer.FilterColumns= 1
sink= diko.framework.output.ArffWriter
```

Abbildung 14.4: Beispielkonfigurationsdatei *properties.txt*

source.jdbcLogin) und das Passwort (Schlüssel *jdbcPassword*) für die Verbindung benötigt. Mit diesen Informationen ist das Framework in der Lage, eine JDBC-Verbindung aufzubauen, um aus einer Datenbank die Quelldaten zu lesen.
Damit die Quelldaten genau spezifiziert sind, wird ein SQL-Statement angegeben, welches die entsprechenden Attribute aus den Tabellen der Quelldatenbank liest. Hierzu dient der Schlüssel *source.jdbcSQL*. Damit das Framework die ausgelesenen Attribute korrekt in WEKA importieren kann, werden Informationen über die Beschaffenheit der gelesenen Attribute benötigt. Diese werden beispielhaft für die beiden ausgelesenen Attribute spezifiziert. Hier ist das erste Attribut die HändlerID, die ein numerisches Datum darstellt und über den Schlüssel *attributeName_X* angegeben wird, wobei das Zeichen X

```
$java diko.framework.AlgorithmRunner properties.txt
@relation clusterResult
@attribute haendlerid numeric
@attribute name {'Haendler 1','Haendler 2'}
@attribute clusterId numeric

@data
1,'Haendler 1',0
2,'Haendler 2',1

kMeans
======

Number of iterations: 2

Cluster centroids:

Cluster 0
   Haendler 1
Cluster 1
   Haendler 2
```

Abbildung 14.5: Ausgabe des Beispiellaufes

durch die entsprechende Nummer des Attributs ersetzt wird. Das zweite ausgelesene Attribut ist der Name des Händlers, eine nominale Zeichenkette.

Um dem Framework mitzuteilen, welcher Analyse-Algorithmus anzuwenden ist, wird nun der Name des entsprechenden Algorithmus im Schlüssel *algorithm* angegeben, in diesem Falle die Java-Klasse *diko.framework.algorithms.SimpleKMeansController*.

Mit diesen Angaben ist die Konfiguration des Frameworks für das anfangs beschriebene Anwendungsszenario abgeschlossen und das Framework kann gestartet werden. Das geschieht wie in Abbildung 14.5 zu sehen über die *main*-Methode der Klasse *AlgorithmRunner*. Diese erwartet als einzigen Parameter den Namen der Datei, die die Parameter zur Steuerung des Frameworks enthält, in diesem Fall *properties.txt* im aktuellen Verzeichnis, deren Inhalt wie beschrieben in Abbildung 14.4 dargestellt ist.

14.5.2 Komplexe Analyse mittels AnalysisRunner

Wie im vorherigen Abschnitt gezeigt, ist es sehr leicht, mit dem Framework einfache Analysen durchzuführen. Es wird lediglich eine Konfigurationsdatei, die das Framework steuert, benötigt, deren Name als Kommandozeilenparameter angegeben wird.
Das Durchführen komplexer Analysen funktioniert ähnlich, nur dass hier $n+1$ Konfigurationsdateien benötigt werden, um n einfache Analysen zu einer komplexen zusammenzufassen. Dabei ist eine Konfigurationsdatei für die Steuerung des Ablaufes der komplexen Analyse zuständig, die übrigen n Dateien steuern die im Kern ablaufenden einfachen Analysen.
Abbildung 14.6 zeigt eine Konfigurationsdatei für eine komplexe Analyse, in der zwei Preprozessoren, eine einfache Analyse und zwei Postprozessoren zu einer komplexen Analyse zusammengefasst werden. Hier werden jeweils zwei Pre- und Postprozessoren definiert, indem

```
analysis_0.properties = properties.txt
analysis_0.preprocessor_0 = diko.personalisierung.\
 cardprovider.preprocessing.BasketClusteringDBTransform
analysis_0.preprocessor_1 = diko.personalisierung.\
 cardprovider.preprocessing.BasketClusteringDBTransform
analysis_0.postprocessor_0 = diko.personalisierung.\
 cardprovider.preprocessing.BasketClusteringDBTransform
analysis_0.postprocessor_1 = diko.personalisierung.\
 cardprovider.preprocessing.BasketClusteringDBTransform
```

Abbildung 14.6: Konfigurationsdatei für eine komplexe Analyse

die vollständigen Klassennamen einer Java-Klasse angegeben werden, die das Interface *PrePostProcessor* implementiert.[2] Der Schlüssel *analysis_0.properties* spezifiziert den Dateinamen der Konfigurationsdatei, die die Parameter für die eigentliche Analyse definiert. In diesem Falle ist es die Datei *properties.txt*, die schon im vorherigen Abschnitt für die Definition einer einfachen Analyse verwendet wurde (Abbildung 14.4, Seite 127). Gestartet wird eine komplexe Analyse

[2]In diesem Fall wird für jeden Pre- und Postprozessor dieselbe Klasse genutzt, was in einem konkreten Anwendungsfall sicher nicht sinnvoll wäre. Hier dient es lediglich dazu, zu zeigen, welche Schlüssel mit welchem Wert belegt werden müssen.

wie diese durch die Klasse *AnalysisRunner*. Die Ausgabe ist in diesem

```
$java diko.framework.AnalysisRunner complexProperties.txt
[...]
```

Abbildung 14.7: Starten einer komplexen Analyse

Fall dieselbe wie in Abbildung 14.5 (Seite 128), da die hier verwendeten Pre- und Postprozessoren keine Ausgabe erzeugen und somit nur die Ergebnisse der eingebetteten einfachen Analyse ausgegeben werden.

14.6 Fazit

Mit dem Framework ist, wie beschrieben, eine Java-Bibliothek entstanden, die es ermöglicht, metadatengesteuerte Analysen auf verschiedensten Datenquellen durchzuführen. Für die Analyse stehen mehrere Clusterer von WEKA, sowie ein von der Projektgruppe selbst erstellter Algorithmus zum Finden von kalenderbasierten Assoziationsregeln zur Verfügung. Damit sind die Anforderungen aus Abschnitt 14.1 und Kapitel 10 erfüllt.
Das Framework leistet sogar mehr als gefordert. Durch die Möglichkeiten der komplexen Analyse (siehe Abschnitte 14.2 und 14.5.2) ist es möglich, in einem Lauf mehrere Analysen zu betreiben und diese mit beliebigen Pre- und Postprozessoren zu verknüpfen.
Allerdings ist das Framework gewissen Einschränkungen unterlegen, die im Kontext der Projektgruppe jedoch nicht relevant sind. Zum Beispiel ist es ohne Änderungen am Framework nicht möglich, Klassifikatoren zur Analyse zu verwenden. Der Grund ist, dass Klassifikatoren zwei unterschiedliche Eingaberelationen benötigen, die Test– und die Trainingsdaten (siehe auch [Pre03]). Das Framework unterstützt allerdings nur eine Eingaberelation pro Analyse.
Weiterhin wäre eine Funktion wünschenswert, die für Algorithmen, die dies konzeptionell unterstützen, eine inkrementelle Analyse ermöglicht. Einige Klassifikatoren in WEKA unterstützen dies beispielsweise bereits. Das Framework kann davon jedoch keinen Gebrauch machen.
Ein wichtiger Punkt ist sicherlich auch die Verwendung von Metadaten-Standards zur Steuerung des Frameworks. Darauf wurde bei

der Implementierung jedoch verzichtet, da die Einarbeitung in einen Metadaten-Standard als zu zeitraubend angesehen wurde. Stattdessen wurde ein eigenes Format entwickelt. Der Nachteil dessen ist die fehlende Interoperabilität mit anderen Metadatenformaten.
Trotz der Unzulänglichkeiten ist das Framework in der Lage, die im Rahmen der Projektgruppe erforderlichen Aufgaben wahrzunehmen und kann somit als erfolgreich angesehen werden.

Teil IV

Personalisierungskonzept

Dieser Teil beschreibt die Personalisierung der Projektgruppe „Personalisierung internetbasierter Handelsszenarien". Personalisierung im Rahmen der Projektgruppe bezeichnet die Anpassung der Webseiten eines Online-Shops an die angenommenen Bedürfnisse der Kunden. Das Ziel dieser Personalisierung ist es, den Kunden eines Shops auf Basis der Auswertung der gesammelten kundenbezogenen Daten relevante und interessante Angebote in einer auf ihn abgestimmten Form zu unterbreiten. Die Aufgabenstellung des Personalisierungskonzeptes innerhalb der Projektgruppe bestand im ersten Schritt darin, ein Konzept zu erstellen, in dem mögliche nicht-temporale Analysen und deren Anwendung (vgl. Kapitel 15 und 16) sowie temporale Analysen und deren Anwendung (vgl. Kapitel 17 und Abschnitt 18) vorgestellt werden. Im Anschluss sollen diese vorgestellten Analysen auf ihre Umsetzbarkeit überprüft und gegebenenfalls auch realisiert werden. Abschließend soll beschrieben werden, wie die Personalisierung im Online-Shop durch Anwendung der Analyseergebnisse erreicht werden kann.

Das Ziel des Personalisierungskonzeptes ist es, die Anwendung der vorhandenen Personalisierungsmöglichkeiten zu erläutern. Dabei werden die zugänglichen Algorithmen, ihre Funktionsweise und ihr Potenzial für den Online-Shop dargestellt. Dieses Dokument soll die einzelnen Anwendungen erörtern und abschließende Empfehlungen für Umsetzung der Personalisierung im Online-Shop geben.

Um die geforderten Ziele zu erreichen, wird untersucht, welche Analysen mit der gegebenen Datenbasis möglich erscheinen und wie eine Anwendung im Online-Shop gestaltet werden kann. Auf Basis dieses Arbeitsschrittes werden folgende Analysemöglichkeiten umgesetzt: Clustering nach Kunden (siehe Abschnitt 15.1.1), Clustering nach Kunden und gekauften Produkten (siehe Abschnitt 16.2) und das Clustering von Warenkörben nach Produkttypen (siehe Abschnitt 16.3) als Vertreter der nicht-temporalen Analysen. Als temporale Analysemöglichkeiten werden kalendarische Muster sowie temporales Clustering umgesetzt (siehe Kapitel 17).

15 Konzept - Nicht-temporale Personalisierung

Dieses Konzept stellt die Gedanken der Projektgruppe über mögliche Analyseverfahren dar, deren Ergebnisse für die Erstellung eines personalisierten Angebotes im Online-Shop verwendet werden könnten. Es soll insbesondere überprüft werden, auf welche Weise die Analyseergebnisse interpretiert werden und in welcher Form sie weiterverwendet werden können.
Um den zu betrachtenden Online-Shop zu individualisieren, sind die vorhandenen Daten des Händler 1 (siehe [iH03] und Teil II) hinsichtlich ihrer möglichen Verwendung zu untersuchen. Es ist zu berücksichtigen, welche Analysen durch die verwendete Bibliothek WEKA [WF01] unterstützt werden und weiterhin welche Analysen mit den als WEKA-Erweiterung entwickelten Analysealgorithmen umgesetzt werden können. Des Weiteren ist zu überlegen, ob die selbst entwickelten Verfahren ausreichend sind oder noch erweitert werden müssen.
Für die nicht-temporale Personalisierung werden verschiedene Formen des Clustering sowohl einzeln als auch in Verbindung mit Assoziationsanalysen vorgestellt.
Bei den beiden vorgestellten Verfahren handelt es sich um das Clustering und um die Assoziationsanalyse, die auf verschiedene Weise eingesetzt werden können. Weiterhin ist realisierbar, einzelne Analysen auch aufeinander aufbauend durchzuführen. So könnte es beispielsweise möglich sein, dass Assoziationsanalysen auf die durch vorheriges Clustering ermittelten Kundengruppen vollzogen werden.
Die Beschreibungen der einzelnen Analysen sind im einzelnen wie folgt aufgebaut:

- **Ausgangsdaten:** Hier werden die zu betrachtenden Relationen und die zu analysierenden Attribute aufgelistet, welche die Basisdaten der Analyse darstellen.
- **Analyseverfahren:** Namen der möglichen Analyseverfahren (Namen der Algorithmen).

- **Interpretation:** Beschreibung der zu erwartenden bzw. gewünschten Ergebnisse und wie diese Daten zu interpretieren sind.

- **Beispiel:** Ein Beispiel zur Verdeutlichung.

- **Anwendung:** Schlußfolgerungen für den konkreten Einsatz im Shop für den Fall, dass die Ergebnisse den Erwartungen entsprechen.

Die Analysen sollen lediglich auf Daten des Händler 1 zugreifen. Die Analyse „fremder“ Daten bietet keinen sichtbaren Mehrwert für die Personalisierung entsprechend der Ziele der Projektgruppe: Bei „fremden“ Daten handelt es sich um Daten, die nicht von Händler 1 stammen. Im Rahmen der Projektgruppe werden demnach die Daten von Händler 2 nicht beachtet. Da es sich bei Händler 2 um einen klassischen Offline-Händler handelt, erscheint es wenig sinnvoll, seine Daten für einen Online-Shop zu verwenden.
Des Weiteren ist zu beachten, dass aus Zeitgründen keine „Ad-hoc“-Analysen durchgeführt werden, d.h. die von einem Kunden gelieferten Daten werden nicht analysiert, während der Kunde in dem Online-Shop eingeloggt ist. Stattdessen werden die Daten zunächst gesammelt, dann analysiert und die daraus resultierenden Ergebnisse zu einem späteren Zeitpunkt für die Personalisierung zur Verfügung gestellt.
Die Analyseverfahren, die im Rahmen der Projektgruppe verwendet werden sollen, werden in Abschnitt 16.1.2 ausführlicher erläutert.

15.1 Einfache Analysen

In diesem Abschnitt werden Analysen beschrieben, die ausschliesslich auf Daten des Händler 1 durchgeführt wurden. Dabei sollten die Analysen sowohl auf der Datenbank des Händler 1 als auch auf der Datenbank des Kartenanbieters funktionieren können. Es werden keine Ergebnisse vorangegangener oder anderer Analysen verwendet. Die Analysealgorithmen sollen eigenständig möglichst verschiedene Gruppen finden, die in sich jedoch möglichst homogen sind.

15.1.1 Clustering nach Kunden

Beim Clustering nach Kunden sollen einzelne Kundengruppen ermittelt werden, deren Mitglieder gemeinsame demographische Daten aufweisen.

Ausgangsdaten: Bezieht sich diese Analyse auf die Datenbank des Händler 1, sind die Ausgangsdaten der Relation *Kunde* des Datenbankschemas von Händler 1 ersichtlich. Mit dem Attribut *gebdatum* wird das Geburtsdatum des Kunden ersichtlich und in dem Attribut *geschlecht* kann festgestellt werden, ob es sich um eine Frau oder Mann handelt. In der Relation *Adresse* wird in dem Attribut *plz* die Postleitzahl des Kunden erkennbar. Für die Analysen können diese Attribute beliebig kombiniert werden.

Analyseverfahren: Clustering (SimpleKMeans, Cobweb, EM, XMeans).

Interpretation: Die Ergebnisse dieser Analysen sollen dazu dienen, die Menge aller Kunden in einzelne Gruppen aufzuteilen wie: „Kunden gleicher Herkunft", „Kunden gleichen Geschlechts" und „Kunden gleichen Alters" sowie Vereinigungen dieser Mengen. Die einzelnen Gruppen stellen Kundengruppen dar.

Beispiel: Mögliche Ergebnisse dieser Analyse können folgende Gruppen sein:

- Kunden aus der Region Ammerland (Clustering nach PLZ)
- weibliche Kunden aus der Region Ammerland (Clustering nach PLZ und Geschlecht)
- weibliche Kunden aus der Region Ammerland zwischen 30-45 Jahren (Clustering nach PLZ, Geschlecht und Geburtsdatum)

Anwendung: Konnte ein Kunde erfolgreich einer Kundengruppe zugeordnet werden, könnten ihm Artikel angeboten werden, die bereits von anderen Kunden der gleichen Gruppe erworben wurden. Die Auswahl dieser Artikel für das personalisierte Angebot kann auf Zufallsbasis erfolgen oder alternativ kann ebenso aus einer Liste der „Top5" der beliebtesten Produkte einer Kundengruppe gewählt werden.
Bei diesem Clustering erfolgt lediglich eine Einteilung der Kunden

in Gruppen anhand der demographischen Kundendaten. Die von den Kunden erworbenen Produkte bleiben jedoch bei der Clusterbildung unberücksichtigt. Die im nächsten Abschnitt vorgestellte Analyse clustert hingegen sowohl nach Kunden als auch nach gekauften Produkten.
Es könnte ebenso sinnvoll sein, diese Clusterergebnisse nicht separat, sondern in Verbindung mit anderen Analysen (z.B. Anwendung der Assoziationsanalyse in Verbindung mit Produkten) zu nutzen. Das Clustering kann damit Grundlage für weitere Analysen sein, auf die an anderer Stelle näher eingegangen wird (siehe Kapitel 15.3.3).
Im Rahmen der Projektarbeit wird das Clustering nach Kunden durchgeführt und deren Ergebnisse mittels einer „Top5“-Liste für ein personalisiertes Angebot verwendet (siehe Abschnitt 16.1.7).

15.1.2 Clustering nach Kunden und gekauften Produkten

Die von dieser Analyse zu findenden Kundengruppen sollen anhand gekaufter bzw. bestellter Waren ermittelt werden. Es existieren verschiedene Granularitätsebenen, von denen aus die Produkte betrachtet werden können. Beispielsweise ist „Milka“ ein Element der Menge „Schokolade“, „Schokolade“ wiederum eine Teilmenge von „Süssigkeiten“ usw.. Es ergeben sich Zusammenhänge wie z.B. „Kunde A kauft Schokolade“ oder „Kunde A kauft Süssigkeiten“, die auf unterschiedlichen Abstraktionsebenen liegen. Die Mitglieder der Kundengruppen sollen ein gemeinsames Kaufverhalten aufweisen.
Wenn auf einer höheren Abstraktionsebene analysiert wird, kann die Clusteranzahl reduziert werden. Denn unterschiedliche Kunden, die zwar auf einer niedrigen Abstraktionsebene verschiedene Produkte kaufen, können trotzdem auf einer höheren Ebene zu einer Gruppe zusammengefasst werden, wenn die verschiedenen Kunden ihre Produkte z.B. hauptsächlich aus der übergeordneten Ebene „Bekleidung“ kaufen.

Ausgangsdaten: Sofern diese Analyse beispielsweise auf den in der Kartenanbieterdatenbank integrierten Daten des Händler 1 basiert, sind die Ausgangsdaten über verschiedene Relationen zu erhalten: Die Identität der Kunden kann über die Relation *Transaktion* über das Attribut *Inhaber* festgestellt werden. Die Anzahl der gekauften Produkte ist über die Relation *Posten* über das Attribut *Anzahl* verfügbar. Über die Relation *Einordnung* wird durch das Attribut

Warengruppeid die Zugehörigkeit eines Produktes zu einer Warengruppe ersichtlich.

Analyseverfahren: Clustering (SimpleKMeans, cobweb, EM, XMeans).

Interpretation: Die zu erwartenden Ergebnnisse sollen dazu dienen, die Menge der Kunden aufgrund ihrer Einkäufe in bestimmte Gruppen zu unterteilen. Vorteilhaft bei diesem Verfahren ist die Berücksichtigung der Produkthierarchien.

Beispiel: Mögliche Ergebnisse dieser Analyse sind folgende Gruppen:

- Kunden, die hauptsächlich Nahrungsmittel kaufen
  - Kunden, die hauptsächlich Süssigkeiten kaufen
  - Kunden, die hauptsächlich Getränke kaufen
    - Kunden, die „Pepsi Light“ kaufen
    - Kunden, die „Coca Cola“ kaufen

Anwendung: Wurde ein Kunde erfolgreich einer wie im vorangegangenen Beispiel beschriebenen Kundengruppe zugeordnet, können ihm direkt Produkte angeboten werden, die von Mitgliedern der gleichen Kundengruppe gekauft wurden.
So kann einem Kunden, der „Schokolade“ kauft bzw. sucht, „Milka“ angeboten werden, wenn die anderen Kunden im selben Cluster überwiegend „Milka“ gekauft haben.
Es könnte ebenso sinnvoll sein, diese Clusterergebnisse nicht separat, sondern in Verbindung mit anderen Analysen (z.B. Anwendung der Assoziationsanalyse in Verbindung mit Produkten) zu nutzen. Das Clustering könnte damit Grundlage für weitere Analysen sein, auf die an anderer Stelle näher eingegangen wird (siehe Kapitel 15.3.3).

15.1.3 Assoziationsanalyse

Unter dem Begriff Assoziationsanalyse wird das Identifizieren und Beschreiben von Regionen bzw. Datenbereichen in einer Datenbank verstanden, in denen mit hoher Wahrscheinlichkeit mehrere Werte

gleichzeitig auftreten. Es wird mit der Assoziationsanalyse nach Assoziationsregeln gesucht, mit Hilfe derer die Ergebnisse dargestellt werden (vgl. [Saa03]). Die Analyseergebnisse können dabei durch Angabe einer minimalen Konfidenz und eines minimalen Supports eingeschränkt werden. Für diese beiden Interessantheitsmaße können Schwellenwerte vorgegeben werden. Der Support gibt prozentual an, in wieviel Transaktionen beide Produkte zusammen erworben werden, in Bezug zu allen Transaktionen. Für die Konfidenz von Produkt A und B wird die Anzahl der Transaktionen bestimmt, die A und B enthalten, in Bezug zu allen Transaktionen, in denen A erfüllt ist. Es wird die Zuverlässigkeit der Regel bewertet, d.h. wie wahrscheinlich der Kauf von Produkt B ist, wenn Produkt A gekauft wurde. Die Konfidenz wird demnach als eine bedingte Wahrscheinlichkeit definiert.

Ausgangsdaten: Wenn die Analyse beispielsweise auf der Datenbank des Händler 1 basieren soll, dann befinden sich die Ausgangsdaten in Relation *Posten*.

Analyseverfahren: Assoziationsanalyse (Apriori).

Interpretation: Aus diesen Analyseergebnissen können Regeln in der Form: „wenn Produkt A gekauft wird, wird ebenfalls Produkt B gekauft“ abgeleitet werden. Mit Hilfe einer Assoziationsanalyse soll ermittelt werden, welche Artikel zusammen gekauft werden.

Beispiel: Es kann in den vorhandenen Daten beispielsweise eine Regel „wenn Bier gekauft wird, werden auch Chips gekauft“ gefunden werden. Für eine derartige Regel kann z.B. ein Support von 10% vorgeschrieben werden; d.h. in mindestens 10% aller Einkäufe müssen „Bier“ und Chips“ vorhanden sein. Zudem soll eine Konfidenz von wenigstens 50% vorliegen, d.h. in mindestens jeder zweiten Transaktion, in der „Bier“ vorkommt, müssen auch „Chips“ vorkommen.

Anwendung: Wurde in den Ausgangsdaten eine Regel gefunden, die den Schwellenwerten genügt, können diese Produkte den Kunden als Produktbundles bzw. Produktpakete angeboten werden. Sobald ein Kunde im Shop ein bestimmtes Produkt kauft, kann ihm das dazugehörige Produkt angepriesen werden. Da die Regeln für die Menge

aller Kunden bestimmt wurden, kann es sein, dass einzelne Kunden ein anderes Kaufverhalten aufweisen (was auch durch Support und Konfidenz wiedergespiegelt wird).
Kombinierte Analysen (siehe Abschnitt 15.2) bieten weitergehende Möglichkeiten. Hierbei können zunächst spezifische Kundengruppen durch das Clustering identifiziert werden, auf denen im Anschluss die Assoziationsanalyse durchgeführt werden kann.
Die in WEKA implementierte nicht-temporale Assoziationsanalyse ist inkompatibel zu der verwendeten Datenstruktur, weil die betrachteten Transaktionen für diesen Algorithmus eine einheitliche Tupelgröße besitzen müssen. Um dieses Problem zu lösen, wäre eine umfangreiche Vorverarbeitung notwendig.

15.2 Kombinierte Analyseverfahren

In diesem Abschnitt werden Analysen beschrieben, die die Ergebnisse von vorherigen Clusteranalysen als Grundlage für eine Assoziationsanalyse verwenden.

15.2.1 Clustering nach Kunden in Verbindung mit einer Assoziationsanalyse

Durch das Clustering nach Kunden werden die Kunden aufgrund demographischer Merkmale (PLZ, Geschlecht, Geburtsdatum) in Gruppen aufgeteilt (siehe Abschnitt 15.1.1). Im Anschluss kann die unter Abschnitt 15.1.3 beschriebene Assoziationsanalyse durchgeführt werden, mit dem Unterschied, dass die Analyse auf die Transaktionsdaten der Kunden eines Clusters beschränkt wird, anstatt wie bei der einfachen Assoziationsanalyse auf die Daten aller Kunden (siehe Abbildung 15.1).

Ausgangsdaten: Die Ausgangsdaten sind Ergebnisse des Clustering nach Kunden und stellen somit spezifische Kundengruppen dar (wie z.B. Kunden aus der Region Ammerland).

Analyseverfahren: Assoziationsanalyse (Apriori).

Interpretation: Die Ergebnisse dieses kombinierten Anaylseverfahrens entsprechen den Ergebnissen der Assoziationsanalyse (siehe Ab-

schnitt 15.1.3). Allerdings werden Regeln nur auf den vorher zugrunde gelegten Kundengruppen gesucht. Durch die Kombination mit der Assoziationsanalyse ist diese Analyse gezielter und es können speziellere Regeln als bei der einfachen Assoziationsanalyse gefunden werden. Allerdings ist die Gültigkeit der gefundenen Regeln auf diese einzelnen Cluster beschränkt.

Beispiel:

- 50% aller Kunden aus der Region Ostfriesland, die Ostfriesentee der Marke Bünting kaufen, kaufen auch Kluntjes

Anwendung: Sobald ein Kunde im Shop einer Kundengruppe zugeordnet werden konnte und ein bestimmtes Produkt kauft, könnte ihm das dazugehörige Produkt angepriesen werden.

15.2.2 Clustering nach Kunden und gekauften Produkten in Verbindung mit einer Assoziationsanalyse

Durch das Clustering nach Kunden und gekauften Produkten werden die Kunden aufgrund ihrer getätigten Einkäufe in Gruppen eingeteilt (vgl. Abschnitt 15.1.2). Im Anschluss könnte die unter Abschnitt 15.1.3 beschriebene Assoziationsanalyse durchgeführt werden (siehe Abbildung 15.2). Hierbei werden die Ausgangsdaten durch das vorherige Clustering reduziert und verfeinert. Zudem wäre es möglich, für die Personalisierung sinnvollere Daten zu finden (im Vergleich zu dem Clustering nach Kunden i.V.m. der Assoziationsanalyse), da nun ebenfalls gekaufte Produkte berücksichtigt würden.

Ausgangsdaten: Die Ausgangsdaten sind Ergebnisse des Clustering nach Kunden und gekauften Produkten und stellen somit spezifische Kundengruppen dar (wie z.B. Kunden, die hauptsächlich Süssigkeiten kaufen).

Analyseverfahren: Assoziationsanalyse (Apriori).

Interpretation: Die Ergebnisse dieses kombinierten Anaylseverfahrens entsprechen den Ergebnissen der Assoziationsanalyse (siehe Abschnitt 15.1.3). Allerdings werden Regeln nur bzgl. der vorher zugrunde gelegten Kundengruppen gesucht.

Beispiel:

- Kunden, die aus der Region Ammerland kommen und die hauptsächlich Nahrungsmittel kaufen, kaufen zu Nudeln mit einer Konfidenz von 63,7% auch eine Nudelsauce mit Hackfleisch.

Anwendung: Die Umsetzung kann wie in Abschnitt 15.1.3 erfolgen und ermöglicht einen höheren Grad an Personalisierung. Die Gültigkeit der gefundenen Regeln ist allerdings auf einzelne Cluster beschränkt.

15.3 Weitere Möglichkeiten des Clusteralgorithmus

Außer den im vorangehenden Text erläuterten Analysen gibt es weitere für die Projektarbeit anwendbare Analyseverfahren. Diese werden im Kontext der Projektgruppe nicht eingesetzt werden, da die temporalen Analysen im Vordergrund stehen.

15.3.1 Clustering nach Produktpreis oder nach Produktkategorie in Verbindung mit dem Preis

Die Ergebnisse dieser Analysen liefern Informationen über die Preisniveaus verschiedener Produkte oder Produktkategorien, die bei einem Clustering ohne Preis nicht ersichtlich wären.

Ausgangsdaten: Wenn die Analyse beispielsweise auf der Datenbank des Händler 1 basieren soll, dann sind die Ausgangsdaten in der Relation *Produkt*, in welcher der Verkaufspreis zu finden ist und in der Relation *Produkt_Kategorie*, in der die Warengruppen des jeweiligen Produktes stehen, ersichtlich.

Analyseverfahren: Clustering (SimpleKMeans, cobweb, EM, XMeans).

Interpretation: Durch das Clustering nach Produktpreis können gekaufte bzw. bestellte Produkte in Preisklassen gruppiert werden. Das Clustering nach Produktkategorie i.V.m. dem Preis bewirkt eine Einteilung der Produktkategorien in verschiedene Preisniveaus (z.B. in günstig und teuer).

Beispiel:

- günstige und teure Produkte (Clustering nach Produktpreis)
- günstige und teure Süßigkeiten (Clustering nach Produktkategorie i.V.m. dem Preis)

Anwendung: Wenn sich ein Kunde für ein Produkt oder eine Produktkategorie eines bestimmten Preisniveaus interessiert, könnten ihm durch die Ergebnisse dieser Analyse Produkte der entsprechenden Preiskategorie angeboten werden.

15.3.2 Clustering nach Produktpreis oder nach Produktkategorie in Verbindung mit dem Preis wiederum in Verbindung mit Clustering nach Kunden und gekauften Produkten

Durch das Clustering nach Produktpreis oder nach Produktkategorie i.V.m. dem Preis wäre es möglich, Produkte in bestimmte Preis- bzw. Produktkategorien einzuteilen. Diese Kategorien können als Basis für ein Clustering der Kunden nach ihren gekauften Produkten genutzt werden (vgl. Abschnitt 15.1.2). Dadurch können Kunden aufgrund ihrer getätigten Transaktionen den Preis- oder Produktkategorien zugeordnet werden, die anhand des Clustering nach Preis bzw. Kategorie gefunden wurden.

Ausgangsdaten: Die Ausgangsdaten stellen die Preis- oder Produktkategorien dar, die durch das Clustering nach Produktpreis oder nach Produktkategorie i.V.m. dem Preis gefunden wurden (siehe Abschnitt 15.3.1).

Analyseverfahren: Clustering (SimpleKMeans, cobweb, EM, XMeans).

Interpretation: Die Analyse dient dazu, einzelne Kunden den gefundenen Preis- und Produktkategorien zuzuordnen. Dadurch kann mittels vorangegangenen Clustering nach dem Preis z.B. festgestellt werden, ob ein Kunde vorangig günstige Einkäufe tätigt oder ob er ein Kunde ist, der eher teure Produkte bevorzugt. Basiert die Analyse auf Daten, die aus dem Clustering nach Produktkategorie resultieren, kann den Kunden zugeordnet werden, ob sie z.B. günstige oder teure Haushaltswaren kaufen.

Beispiel:

- ein Kunde, der hauptsächlich teure Schokolade kauft, kauft ebenfalls teure Bonbons (Clustering nach Produktpreis in Verbindung mit Clustering nach Kunden und gekauften Produkten)

- ein Kunde, der hauptsächlich teure Nahrungsmittel kauft, kauft jedoch günstige Bekleidung (Clustering nach Produktkategorie in Verbindung mit dem Preis wiederum in Verbindung mit Clustering nach Kunden und gekauften Produkten)

Anwendung: Durch die Kombination dieser beiden Verfahren können die Kunden in speziellere Gruppen eingeteilt werden als beim einfachen Clustering nach Kunden und gekauften Produkten, da der Preis der Produkte mit einbezogen wird. Die Umsetzung könnte wie in Abschnitt 15.1.2 erfolgen.

15.3.3 Zweifaches Clustering nach Kunden

Eine weitere Möglichkeit besteht darin, auf den Ergebnissen des Clustering ein weiteres Clustering durchzuführen.

Ausgangsdaten: Die Ausgangsdaten stellen die gefundenen Kundengruppen dar, die aus dem ersten Clustering nach Kunden stammen (siehe Abschnitt 15.1.1).

Analyseverfahren: Clustering (SimpleKMeans, cobweb, EM, XMeans).

Interpretation: Im Vergleich zum Clustering von Kunden können noch detailliertere Kundengruppen bestimmt werden, da das zweite Clustering auf bereits vorgruppierten Kundendaten operiert.

Beispiel:

- weibliche Kunden aus Oldenburg (1. Clustering nach Kunden) im Alter zwischen 27 und 34 (2. Clustering nach Kunden)

Anwendung: Wenn ein Kunde einer Kundengruppe zugeordnet werden würde, könnten ihm Artikel angeboten werden, die bereits von anderen Kunden der gleichen Gruppe erworben wurden. Durch das zweifache Clustering können lediglich genauere Kundengruppen gefunden werden; die Umsetzung entspricht daher der des einfachen Clustering nach Kunden (siehe Abschnitt 15.1.1).

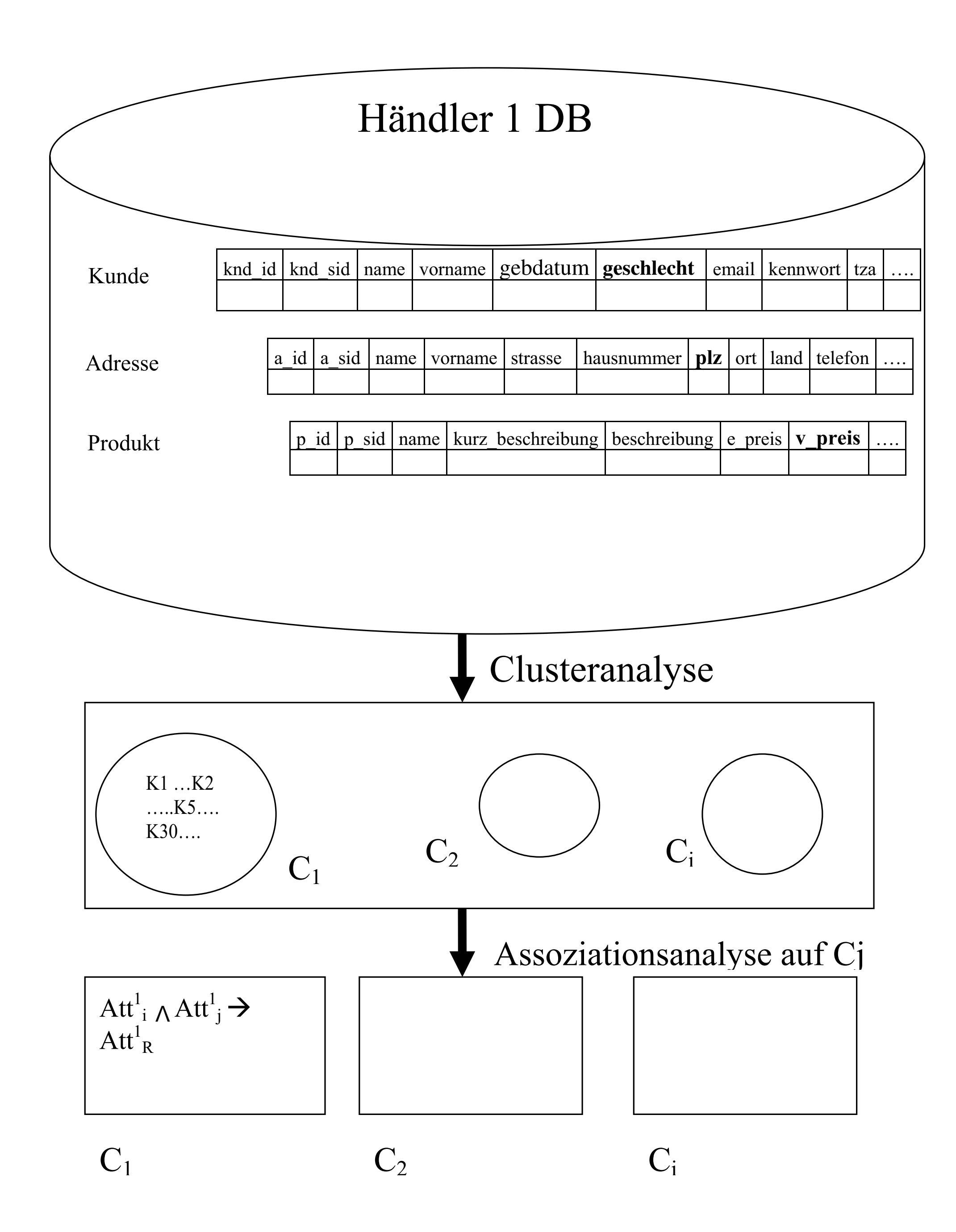


Abbildung 15.1: Clustering nach Kunden i.V.m. Assoziationsanalyse

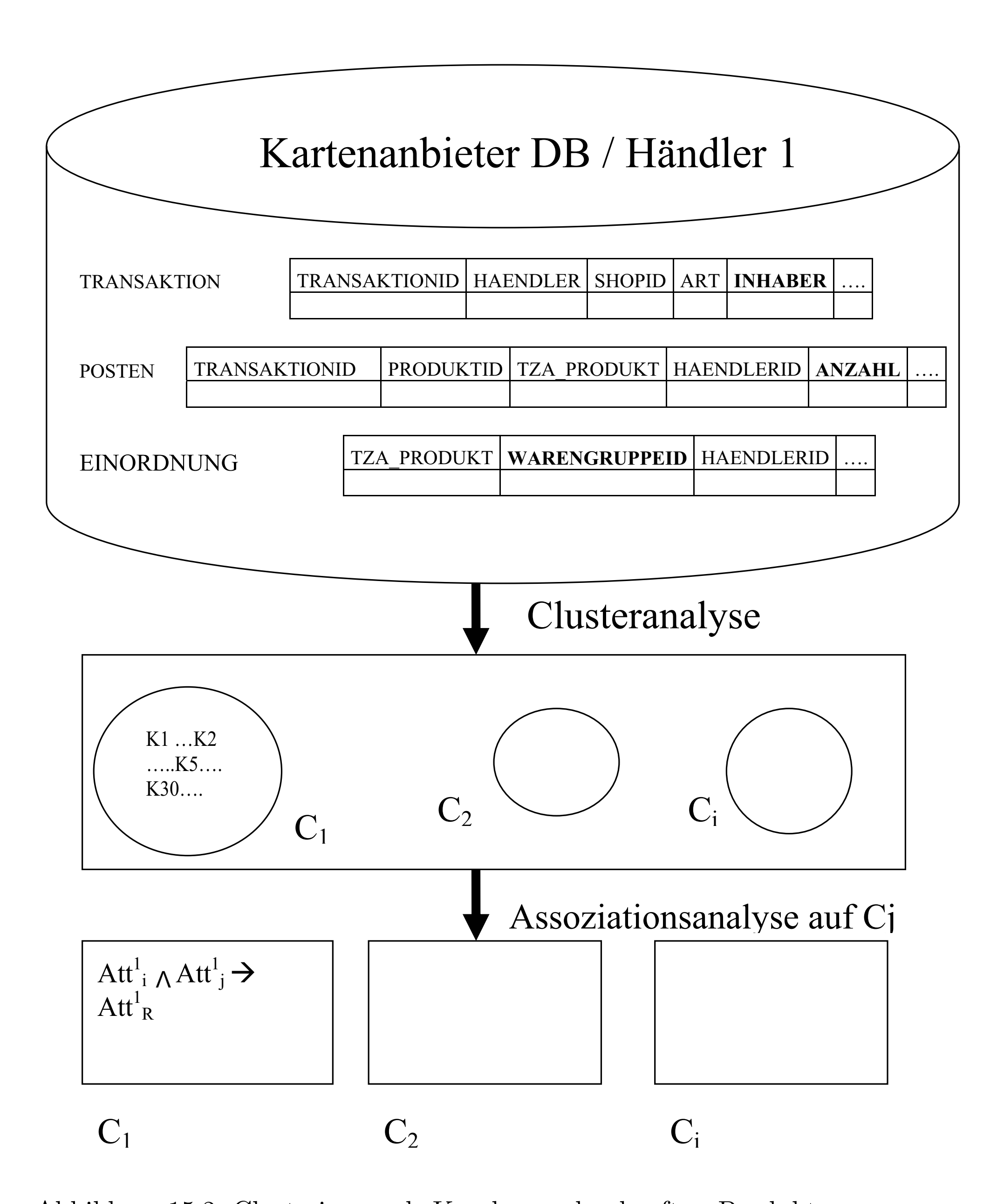


Abbildung 15.2: Clustering nach Kunden und gekauften Produkten i.V.m. Assoziationsanalyse

16 Anwendung - Nicht-temporale Analysen

Von den in dem Konzept vorgestellten Möglichkeiten der Analysen wurden nicht alle genannten Beispiele umgesetzt. Dies resultiert in erster Linie daraus, dass der Schwerpunkt der Projektarbeit auf den temporalen Analysen liegt.
Aus der Reihe der einfachen Analysen wurde die Clustering nach Kunden umgesetzt, die in Abschnitt 16.1 detailliert beschrieben wurde. Das Clustering nach Kunden und gekauften Produkten ist nicht zur Anwendung gekommen, da die Ergebnisse dieses Clusterings im Rahmen der Projektarbeit im Online-Shop aufgrund der mangelnden Qualität der Testdaten nicht umgesetzt werden (siehe Abschnitt 16.2). Somit können die erzeugten Cluster dieser Analyse nicht als Basis für weitere Analysen oder für eine Assoziationsanalyse verwendet werden. Des Weiteren wurde auf Anraten der Projektgruppenleiter die Verwendung der Assoziationsanalyse verworfen, da die Implementierung einer geeigneten zusätzlichen Datenvorverarbeitung über den Rahmen der Projektgruppe hinaus ginge (siehe Abschnitt 15.1.3).
Von den kombinierten Verfahren wurden keine Analysen umgesetzt, da die grundlegenden Analysen nicht realisiert wurden, auf welche die kombinierten Analysen aufbauen. Die nicht-temporale Assoziationsanalyse wurde z.B. im Online-Shop nicht umgesetzt, so dass ebenfalls das Clustering nach Kunden i.V.m einer Assoziationsanalyse nicht angewendet wurde. Aus dem selben Grund wird das Clustering nach Kunden und gekauften Produkten i.V.m. einer Assoziationsanalyse nicht angewendet.
Weiterhin wurde das Clustering nach Produktpreis oder nach Produktkategorie i.V.m. dem Preis aufgrund der Schwerpunktlage auf temporale Analysen nicht weiter verfolgt. Da das Clustering nach Kunden und gekauften Produkten im Rahmen der Projektarbeit ebenfalls aus zeitlichen Gründen nicht im Online-Shop umgesetzt wird, wird auch das Clustering nach Produktpreis oder nach Produktkategorie i.V.m. dem Preis wiederum i.V.m. dem Clustering nach Kun-

den und gekauften Produkten nicht durchgeführt. In Bezug auf das zweifache Clustering nach Kunden ist aufzuführen, dass durch das einfache Clustering nach Kundengruppen bereits Kundengruppen gefunden werden, die für Analyseziele im Rahmen des Projektes ausreichend sind. Daher wird auf diese Analyse verzichtet.

16.1 Clusteranalyse nach Kunden

16.1.1 Einleitung

In diesem Abschnitt wird die Umsetzung des Vorschlages aus dem Konzept der Personalisierung (siehe Kapitel 15) bezüglich des Clusterings auf Kundenattributen beschrieben. Das Clustering ist dabei zu Testzwecken zunächst auf verschiedenen Relationen der Kartenanbieterdatenbank siehe Zwischenbericht [iH03]) durchgeführt worden. Abgesehen von dieser Ausnahme, bleiben alle Analysen auf die Datenbank des Händler 1 beschränkt. Da ein verteiltes Analysesystem vorliegt, können die Analysen problemlos auf diese Weise durchgeführt werden. Neben den verschiedenen Clusterings und deren Ergebnissen, enthält dieser Abschnitt weiterhin eine konkretisierte Beschreibung der endgültigen Anwendung der Analyseergebnisse im Online-Shop. Abgeschlossen wird dieser Abschnitt mit dem technischen Handbuch, in dem die Umsetzung der Implementierung beschrieben wird.

16.1.2 Kennenlernen von WEKA

WEKA[WF01] ist eine Java-Bibliothek, die zum Data Mining verwendet wird. Im Folgenden werden die für das Clustering notwendigen Funktionen und die durchgeführten Testläufe zum Kennenlernen von WEKA vorgestellt.

Funktionen

Die für die Projektgruppe interessanten Funktionen gliedern sich in:

- Vorverarbeitung: In der Phase der Vorverarbeitung gibt es verschiedene Möglichkeiten, die für das Clustering notwendigen Attribute in WEKA zu importieren. Die erste Möglichkeit besteht darin, eine schon vorher angelegte Datei (ARFF-Datei) mit den gewünschten Attributen zu öffnen. Desweiteren läßt sich unter dem Punkt OpenDB direkt eine SQL-Abfrage angeben, welches

aus einer vorher zu definierenden DB-URL ausgeführt wird. Auf diese Weise werden die für die folgende Analyse notwendigen Attribute zur Verfügung gestellt. WEKA bietet dabei die Data Mining-Verfahren Klassifikation, Clustering und Assoziationsanalyse an, wobei in diesem Abschnitt nur das Clustering näher betrachtet wird.

- Clustering: Bei der Auswahl der Clusteralgorithmen lassen sich weitere Voreinstellungen treffen, wobei drei verschiedene zur Auswahl stehen: SimpleKMeans, cobweb und EM. Die im nächsten Punkt erläuterten Testläufe von WEKA haben weiterhin gezeigt, dass der SimpleKMeans über eine übersichtliche Anzahl an zu variierenden Parametern und über eine gut strukturierte Ergebnisausgabe verfügt. Deswegen ist dieser Algorithmus bei den folgenden Analysen den anderen beiden Algorithmen vorgezogen worden.
  SimpleKMeans bietet dem Anwender die Möglichkeit, die Anzahl der zu findenden Cluster zu bestimmen. Eine weitere Stellschraube liegt in der Zahl Seed, welche die zur Initialisierung des Zufallszahlengenerators verwendete Zahl darstellt.

- Visualisierung: Hinsichtlich der Visualisierbarkeit der Analyseergebnisse bietet WEKA die Möglichkeit, Ergebnisse visuell darzustellen. Diese Möglichkeit liefert jedoch für die Analyse nicht zu verwendende Ergebnisse.

Testläufe

Um die Eigenschaften der Clusteralgorithmen von WEKA näher kennenzulernen, ist auf der Kartenanbieterdatenbank eine Entität *Vorname_Test* angelegt worden, die nur das Attribut *Vorname* enthält. Die Instanz des Attributes *Vorname* setzt sich aus den Vornamen *Christian*, *Julia* und *Tim* zusammen, die jeweils vier mal vorkommen. Auf dieser Entität sind nun die ersten Testläufe durchgeführt worden, deren Ergebnisse im Folgenden dargestellt werden:

1. Testlauf Das Ziel des ersten Testlaufes ist es gewesen, drei Cluster zu finden, die jeweils vier Vornamen (*Christian*, *Julia* oder *Tim*) enthalten, da dieses intuitiv sinnvoll erscheint. Die Einstellungen sind Tabelle 16.1 zu entnehmen, wobei Tabelle 16.2 die Ergebnisse enthält.

SQL-Query	select *Vorname* from *Vorname-Test*
Clusteralgorithmus	SimpleKMeans
Seed	1
Anzahl vorgegebener Cluster	3

Tabelle 16.1: Einstellungen 1. Testlauf

Cluster	Centroid	Inhalt/Einträge
0	Julia	8
1	Christian	4

Tabelle 16.2: Ergebnis 1. Testlauf

Ergebnis des ersten Testlaufes Der erste Testlauf hat nicht das zu erwartende Ergebnis geliefert, da nur 2 statt 3 Cluster gefunden worden sind, wobei Cluster 0 sowohl alle Werte für *Julia*, als auch für *Tim* enthält. Lediglich der Cluster 1 enthält die erwarteten 4 Einträge.

2. Testlauf Da der erste Testlauf nicht die intuitiv erwarteten Ergebnisse geliefert hat, ist beim zweiten Testlauf die Variable Seed verändert worden, weil davon ausgegangen werden kann, dass die Clusteranzahl von 3 richtig gewählt worden ist. Das Clustering mit den Einstellungen von Tabelle 16.3 hat schließlich die zu erwartende Clusteranzahl von 3 geliefert, wie es auch in der Ergebnistabelle 16.4 zu sehen ist.

SQL-Query	select *Vorname* from *Vorname-Test*
Clusteralgorithmus	SimpleKMeans
Seed	10
Anzahl vorgegebener Cluster	3

Tabelle 16.3: Einstellungen 2. Testlauf

Cluster	Centroid	Inhalt/Einträge
0	Julia	4
1	Christian	4
2	Tim	4

Tabelle 16.4: Ergebnis 2. Testlauf

Ergebnis des zweiten Testlaufes Der zweite Testlauf hat die vorher erwarteten Ergebnisse geliefert. Damit ist es gelungen, ein erstes Verständnis für die Vorgehensweise des SimpleKMeans Clusteralgorithmus unter WEKA zu erlangen.

3. Testlauf Um einen weiteren Clusteralgorithmus zu testen, ist im Folgenden auf den gleichen Testdaten, die auch schon in den ersten beiden Testläufen verwendet worden sind, der EM-Clusteralgorithmus angewendet worden. Bei diesem Clusteralgorithmus existiert unter WEKA die Möglichkeit, die optimale Clusteranzahl vom Algorithmus automatisch vorgeben zu lassen. Ein Clustering mit diesen Einstellungen auf der Testdatenmenge hat ca. 400 Cluster ergeben, wobei alle 12 Einträge nur einem Cluster zugeordnet worden sind. Auch ein Testlauf mit manuell eingestellter Clusteranzahl von 3 hat nicht die erwünschten Ergebnisse erbracht. Auf Grund dieser schlecht zu interpretierenden Ergebnisse ist dieser Algorithmus im weiteren Verlauf nicht zur Anwendung gekommen. Auch der von WEKA angebotene cobweb-Algorithmus ist während des weiteren Clusterings nicht benutzt worden, da die Anforderung des cobweb-Algorithmus an die Datenhaltung bei großen Datenmengen nicht mit der von der Projektgruppe benutzten, modifizierten Version von WEKA vereinbar ist (siehe Abschnitt 12.3). Aufgrund der Ergebnisse der beschriebenen Testläufe ist entschieden worden, im Folgenden den SimpleKMeans Clusteralgorithmus zu verwenden.

SQL-Query	select *Name*, *E-Preis* from *Produkt*
Clusteralgorithmus	SimpleKMeans
Seed	10
Anzahl vorgegebener Cluster	5

Tabelle 16.5: Einstellungen Clustering Produkte/Preise

Cluster	Centroid	Inhalt/Einträge
0	Produkt 2611 / 106.657	4
1	Die Firma / 25.246	4
2	Produkt 2615 / 146.744	4
3	Produkt 2607 / 66.154	4
4	Produkt 2609 / 185.257	4

Tabelle 16.6: Ergebnis Clustering Produkte/Preise

16.1.3 Clustering auf den ursprünglichen Testdaten

Nachdem die Funktionsweisen der Clusteralgorithmen unter WEKA folglich bekannt gewesen sind, ist damit begonnen worden, Cluster innerhalb der ursprünglich angelegten Testdaten von Händler 1 zu finden. Aus oben bereits aufgeführten Gründen ist dabei ausschließlich der k-Means Algorithmus zur Anwendung gekommen.

Clustering nach Produkten und Preisen

Das erste Ziel ist es gewesen, Cluster zu finden, in denen die Produkte in unterschiedliche Preiskategorien eingeordnet werden. Ein Clustering mit den aus Tabelle 16.5 ablesbaren Werten hat dabei jedoch wenig aussagekräftige Ergebnisse geliefert (vgl. Tabelle 16.6). Eine nähere Betrachtung der Daten des Attributes *Name* der Entität *Produkt* hat schließlich ergeben, dass die Namensdaten für die Produkte durchnummeriert sind. Sie sind daher nicht für ein Clustering geeignet, da innerhalb der Produktnamen keine Ähnlichkeiten auftreten.

Clustering nach Namen und Geburtsdatum

Im Folgenden ist es das Ziel gewesen, Cluster aufgrund des Namens und des Geburtsdatums der Kunden zu finden. Es sollten auf diese Weise Kundengruppen identifiziert werden, die zum Beispiel in die Kategorie Kinder fallen (0-18 Jahre). Dieses ist jedoch nicht möglich gewesen, da die Namen der Kunden in den angelegten Testdaten wiederum durchnummeriert gewesen sind, welches zu nicht aussagekräftigen Ergebnissen geführt hätte.

Qualität der bisher angelegten Testdaten

Aufgrund der negativen Ergebnisse der ersten Clusterversuche auf den bisherigen Testdaten sind diese im Folgenden näher betrachtet worden, wobei mit Hilfe des Oracle Enterprise Managers speziell die Instanzen näher untersucht worden sind. Es hat sich dabei ergeben, dass die angestrebten Kundencluster mit Bezug auf das Geburtsjahr, die Postleitzahl und das Geschlecht nicht realisiert werden können. Der Grund dafür sind die für ein sinnvolles Clustering ungenügenden Testdaten, teilweise aber auch Formate, mit denen keine aussagekräftigen Cluster entstehen bzw. ein Clustering gar nicht möglich ist.

16.1.4 Anlegen und Erweitern von synthetischen Testdaten

Aufgrund der schlechten Verwendbarkeit der bisherigen Testdaten sind im Folgenden synthetische Testdaten für die Attribute *KundenID*, *Geschlecht*, *Geburtsdatum* und *PLZ* angelegt worden. Es ist damit begonnen worden, Kunden mit vier verschiedenen Geburtsjahren, jeweils dem Geschlecht und zwei verschiedenen Postleitzahlen anzulegen, wobei diese im Laufe des Clusterings immer weiter abgeändert worden sind. Auf diesen Daten sind verschiedene Clusterings ausgeführt worden. Die synthetisch angelegten Testdaten umfassen dabei am Ende der hier beschriebenen Erweiterung lediglich 183 Kundendatensätze, um eine leichte Interpretierbarkeit der Analyseergebnisse zu gewährleisten.

1. Erweiterungsschritt

Um sicherzustellen, dass der SimpleKMeans Clusteralgorithmus im Bezug auf das Geburtsjahr sinnvolle Ergebnisse liefert, ist damit begonnen worden, nur nach dem Geburtsjahr der Kunden zu clustern.

SQL-Query	select *GEBURTSDATUM* from *Test_Kunden*
Clusteralgorithmus	SimpleKMeans
Seed	10
Anzahl vorgegebener Cluster	5

Tabelle 16.7: Einstellungen 1. Clustering

Cluster	Centroid	Inhalt/Einträge
0	1943	4
1	1965	4
2	1978	4
2	1990	4

Tabelle 16.8: Ergebnis 1. Clustering

Dabei sind verschiedene Einstellungen der Parameter Seed und Anzahl der Cluster zur Anwendung gekommen. Die ersten getesteten Parametereinstellungen sind der Tabelle 16.7 zu entnehmen, während die Ergebnisse dieses Clusterings in der Abbildung 16.8 abzulesen sind. Um andere Einstellungen zu testen, sind weitere Parametervariationen zur Anwendung gekommen. Dabei haben sowohl die Einstellungen Clusteranzahl 5, Seed 1 und Clusteranzahl 4, Seed -1 identische Ergebnisse beim Clustering ergeben. Die vier unterschiedlichen Geburtsjahre sind vier verschiedenen Clustern zugeordnet worden, wobei die Centroiden jeweils exakt den Jahreszahlen der angelegten Testdaten entsprechen.

2. Erweiterungsschritt

Im Folgenden ist es das Ziel gewesen, zu testen, ob ebenfalls vier Cluster im Bezug auf das Geburtsjahr gebildet werden, wenn Jahreszahlen existieren, die nicht exakt mit den vier bisher angelegten Geburtsjahren übereinstimmen. Es sind daher weitere Kunden mit den vier bisher existierenden Geburtsjahren angelegt worden. Als Folge davon existieren jedoch auch Kunden, die im geringen Maße von diesen vier Geburtsjahren abweichen. Das Clustering mittels des SimpleKMeans Algorithmus hat dabei nach wenigen Versuchen die abweichenden Ge-

burtsjahre den vier schon existierenden Geburtsjahren zugeordnet.

3. Erweiterungsschritt

Aufgrund der beschriebenen Ergebnisse, ist es das neue Ziel gewesen, die Geburtsjahre der Kunden weiter zu differenzieren, um so die synthetisch angelegten Testdaten realitätsnah zu gestalten. Des Weiteren ist die Anzahl der Postleitzahlen von nur zwei verschiedenen Einträgen um weitere ergänzt worden. Es soll auf diese Weise möglich sein, nicht nur Kunden aus zwei verschiedenen Städten zu clustern, sondern auch Kundengruppen aus der entsprechenden Region zu identifizieren. Eine Gesamtübersicht der Testdaten lässt sich in den Tabellen 16.9, 16.10, 16.11, 16.12, 16.13 finden. Sie bilden in dieser Zusammensetzung Kunden der Geburtsjahre 1937 bis 1991 mit zufälliger Häufigkeitsverteilung ab. Des Weiteren ist das Geschlecht dieser Kunden (weiblich=0, männlich=1), sowie die Postleitzahl des jeweiligen Wohnortes der Kunden abgebildet.

16.1.5 Clustering der synthetischen Testdaten

Auf den synthetischen Testdaten sind im Folgenden mehrere Clusterings durchgeführt worden, die in diesem Abschnitt näher beschrieben werden.

Clustering nach Geburtsdatum und Geschlecht

Das Ziel dieser Clusteranaylse ist es gewesen, Kundengruppen im Bezug auf das Geburtsdatum und das dazugehörige Geschlecht zu identifizieren. Die Einträge des Attributes *KundenID* sind bei dieser Analyse und bei den weiteren Analysen durch eine manuell eingestellte Funktion in WEKA ignoriert worden. Die KundenID wird jedoch bei der Ergebnisausgabe angezeigt, um eine Zuordnung der Kunden der jeweiligen Cluster zu gewährleisten. Innerhalb dieses ersten Clusterings sind verschiedene Parametereinstellungen getestet worden, wobei im Folgenden nur die aussagekräftigsten Einstellungen und Ergebnisse dargestellt werden.

1. Clustering Bei der manuellen Durchsicht der synthetisch erzeugten Testdaten sind vier Jahresschwerpunkte (Häufung von Geburtsjahren der Kunden) entdeckt worden. Das Ziel des Clusterings ist dabei das Finden von Clustern gewesen, die im Bezug auf die synthetisch erzeugten Testdaten leicht zu interpretieren sind. Aufgrund der Verteilung der Kundendaten (vgl. die Tabellen 16.9, 16.10, 16.11, 16.12, 16.13) sollen dabei die acht folgenden Cluster bzw. Kundengruppen identifiziert werden:

- 1. männliche erwachsene Kunden um 40 Jahre.
- 2. weibliche erwachsene Kunden um 40 Jahre.
- 3. männliche Frührentner oder ältere Kunden.
- 4. weiblich Frührentner oder ältere Kunden.
- 5. männliche Twens
- 6. weibliche Twens
- 7. männliche jugendliche Kunden
- 8. weibliche jugendliche Kunden

KundenID	*Geschlecht*	*Geburtsdatum*	*PLZ*
1	1	1978	23456
2	0	1965	23123
3	0	1965	23566
4	0	1965	23123
5	0	1965	66345
6	0	1965	66666
7	0	1978	66666
8	1	1978	23123
9	1	1978	23123
10	1	1978	66323
11	1	1978	23123
12	1	1965	66666
13	0	1965	23123
14	0	1965	23123
15	0	1978	66666
16	0	1978	23123
17	0	1965	66789
18	1	1965	23123
19	1	1965	66666
20	1	1978	23777
21	1	1978	23123
22	1	1978	23123
23	1	1978	66666
24	1	1965	23899
25	1	1965	66666
26	0	1978	66666
27	0	1978	23123
28	0	1978	23123
29	1	1990	23223
30	0	1990	66666
31	1	1990	23323
32	1	1990	23344
33	0	1990	66666
34	1	1990	23123
35	1	1990	23123
36	0	1990	66345
37	1	1990	66666
38	1	1990	66457
39	1	1990	66666
40	0	1990	23123

Tabelle 16.9: synthetische Testdaten

KundenID	*Geschlecht*	*Geburtsdatum*	*PLZ*
41	0	1990	23522
42	0	1990	23123
43	0	1990	66522
44	1	1990	66666
45	0	1990	23123
46	1	1990	23123
47	1	1990	66999
48	0	1990	23123
49	0	1990	23001
50	1	1990	66666
51	0	1990	23123
52	1	1990	22990
53	0	1990	23123
54	0	1990	23567
55	0	1990	66666
56	1	1990	66666
57	1	1943	66666
58	1	1943	23123
59	1	1943	22105
60	0	1943	23123
61	0	1943	66666
62	1	1943	23123
63	0	1943	23123
64	0	1943	66777
65	0	1943	66666
66	1	1943	23123
67	1	1943	24102
68	1	1943	23123
69	1	1943	23123
70	0	1943	24203
71	0	1943	66666
72	0	1943	66234
73	0	1943	66211
74	0	1943	23567
75	0	1943	23123
76	1	1943	23123
77	1	1943	66666
78	1	1943	23432
79	0	1943	66666
80	1	1943	23123

Tabelle 16.10: synthetische Testdaten

KundenID	*Geschlecht*	*Geburtsdatum*	*PLZ*
81	1	1991	23123
82	1	1991	66543
83	1	1991	23123
84	0	1989	66666
85	0	1989	24568
86	1	1989	23123
87	1	1988	22999
88	1	1992	23123
89	0	1987	22789
90	0	1987	66333
91	0	1993	66666
92	1	1993	66666
93	1	1986	66666
94	0	1986	66777
95	1	1985	66666
96	1	1985	66666
97	0	1985	66666
98	0	1984	23123
99	0	1983	66666
100	1	1982	23456
101	1	1982	23123
102	1	1981	66666
103	1	1981	23123
104	1	1980	23123
105	0	1980	66323
106	0	1980	23123
107	0	1979	23123
108	0	1979	24679
109	1	1979	66666
110	1	1979	23123
111	1	1977	23123
112	1	1977	23333
113	0	1977	23123
114	0	1977	23555
115	1	1976	66666
116	1	1976	67001
117	1	1975	66666
119	0	1974	23123
120	1	1973	23434

Tabelle 16.11: synthetische Testdaten

KundenID	*Geschlecht*	*Geburtsdatum*	*PLZ*
121	1	1972	65987
122	0	1972	66666
123	1	1971	23123
124	0	1970	23678
125	1	1970	23123
126	1	1969	23999
127	1	1968	66111
128	0	1968	66666
129	0	1967	23123
130	1	1967	23123
131	1	1967	66666
132	0	1966	66234
133	0	1966	23111
134	1	1966	23123
135	1	1966	23562
136	1	1964	66666
137	0	1964	66666
138	1	1963	23123
139	1	1963	23123
140	0	1963	66890
141	0	1962	66666
142	0	1962	23123
143	0	1962	23123
144	0	1961	23123
145	1	1960	23123
146	1	1960	23123
147	1	1959	23123
148	0	1958	65234
149	0	1958	66666
150	0	1957	66345
151	0	1956	66666
152	1	1956	66666
153	1	1955	23123
154	1	1954	23123
155	0	1953	23489
156	1	1952	66212
157	1	1951	66666
158	1	1951	23123
159	1	1950	23123
160	1	1950	23123

Tabelle 16.12: synthetische Testdaten

KundenID	*Geschlecht*	*Geburtsdatum*	*PLZ*
161	0	1950	66345
162	0	1949	66666
163	0	1948	23777
164	0	1947	23123
165	0	1947	66666
166	0	1946	23499
167	0	1946	23123
168	0	1945	66222
169	1	1945	23123
170	1	1945	23566
171	1	1945	66666
172	1	1944	23123
173	1	1944	23123
174	1	1942	23211
175	1	1942	23123
176	0	1942	66789
177	0	1941	66666
178	0	1941	66666
179	0	1940	23345
180	0	1939	23123
181	0	1938	23678
182	0	1938	66999
183	0	1937	23123

Tabelle 16.13: synthetische Testdaten

SQL-Query	select *KundenID*,*Geburtsdatum*, *Geschlecht* from *Test_Kunden*
Clusteralgorithmus	SimpleKMeans
Seed	1
Anzahl vorgegebener Cluster	10

Tabelle 16.14: Einstellungen 1. Clustering

Cluster	Centroid	Inhalt/Einträge
0	1941 / 0	21
1	1963 / 0	22
2	1989 / 0	22
3	1949 / 0	9
4	1977 / 0	15
5	1946 / 1	28
6	1971 / 1	38
7	1988 / 1	28

Tabelle 16.15: Ergebnis 1. Clustering

Da die bisherigen Erfahrungen mit dem k-Means Clusteralgorithmus gezeigt haben, dass die Parametereinstellung für die Anzahl der auszugebenden Cluster etwas höher zu wählen ist als die Zahl der zu erwarteten Cluster, sind die Einstellungen aus Tabelle 16.14 zur Anwendung gekommen.

Ergebnis der ersten Clusterings Die Ergebnisse aus Tabelle 16.15 stellen die Verteilung der Kundendaten nicht entsprechend der oben beschriebenen Interpretierbarkeit dar. So weicht beispielsweise die Jahreszahl aus Cluster 6 sehr stark von dem gefundenen Jahresschwerpunkt ab, der in der Nähe von 1978 liegen sollte (Häufung von Kunden(15 Einträge) mit dem Geburtsjahr 1978). Bei der Untersuchung der Testdaten ist zudem kein Schwerpunkt an männlichen Kunden gefunden worden, die um das Jahr 1971 geboren worden sind. Des weiteren sind zwei Cluster (0 und 3) gefunden worden, die weibliche Kunden mit einem Geburtsdatum aus den fünfziger Jahren beinhalten. Bei Durchsicht der Testdaten sind in diesem Bereich allerdings keine auffälligen Häufigkeiten zu finden gewesen. Aufgrund der Ab-

SQL-Query	select *KundenID*, *Geburtsdatum*, *Geschlecht* from *Test_Kunden*
Clusteralgorithmus	SimpleKMeans
Seed	1
Anzahl vorgegebener Cluster	8

Tabelle 16.16: Einstellungen 2. Clustering

Cluster	Centroid	Inhalt/Einträge
0	1943 / 0	28
1	1977 / 0	15
2	1988 / 0	23
3	1988 / 1	28
4	1962 / 0	23
5	1946 / 1	28
6	1971 / 1	38

Tabelle 16.17: Ergebnis 2. Clustering

weichung von den angestrebten Clusterergebnissen sind im Folgenden weitere Parametereinstellungen getestet worden.

2. Clustering Aufgrund der nicht zufriedenstellenden Ergebnisse des ersten Clusterings sind nun die in Tabelle 16.16 abgebildeten Parametereinstellungen zur Anwendung gekommen.

Ergebnis des zweiten Clusterings Die in Tabelle 16.17 dargestellten Ergebnisse geben die Verteilung der Kunden im Bezug auf die geclusterten Ergebnisse ebenfalls nicht entsprechend der gewünschten Interpretierbarkeit wieder. Wie zu sehen, sind bei diesen Parametereinstellungen lediglich sieben Cluster gefunden worden. Cluster 6 beinhaltet dabei 38 Kunden mit Geburtsjahren um 1971 herum. Bei einem Blick in die Testdaten ist diese Häufung allerdings nicht zu erkennen. Vielmehr liegt die Jahreszahl 1971 zwischen den zwei vermuteten Schwerpunktjahren. Auch das zweite Clustering hat demnach keine Cluster erstellt, die entsprechend der Aufzählung aus Abschnitt 16.1.5 interpretiert werden können.

Weitere Clusterings In der nächsten Phase des Clusterings ist mit einer weiten Streuung des Seed-Parameters (-1000 bis 10000) gearbeitet worden, wobei die Clusteranzahl jeweils bei ca. 8 gelegen hat. Dabei ist wiederum häufig das Problem aufgetreten, dass die gefundenen Cluster nicht die Ergebnisse wiedergegeben haben, die bezüglich der leichten Interpretierbarkeit der Clusterergebnisse wünschenswert gewesen sind. Begründen lässt sich dieses häufig mit der nicht entsprechend gefundenen Anzahl an Clustern, sowie einer abweichenden Zuordnung der Kunden zu diesen Clustern. Die Tabelle 16.18 enthält dabei alle weiteren durchgeführten Clusterings. Als sinnvolle Ergebnisse innerhalb der Tabelle werden alle Clusterergebnisse bezeichnet, die die Verteilung der Kundenattribute entsprechend der gewünschten leichten Interpretation wiedergeben. Existiert in den Ergebnissen eine geringe Abweichung, aber dennoch die Möglichkeit einer sinnvollen Interpretation, so werden die Ergebnisse in der Tabelle ebenfalls noch als sinnvoll bezeichnet. Diese Abweichungen können sein:

- Jahresschwerpunkte in Verbindung mit doppelten Geschlechtszuordnungen
- falsche Jahresschwerpunkte (rechnerisch nicht sinnvoll)
- drei gleiche Jahresschwerpunkte mit den dazugehörigen Geschlecht

Finale Clusterings Nach den vielfältigen Clusterings mit verschiedenen Parametereinstellungen unter WEKA ist letztendlich ein Clustering mit den Einstellungen aus Tabelle 16.19 durchgeführt worden.

Ergebnis des finalen Clusterings Die in Tabelle 16.20 dargestellten Ergebnisse liefern Cluster, die eine Interpretation entsprechend der Aufzählung aus Abschnitt 16.1.5 ermöglichen. Das Clustering hat dabei diese Kundengruppen mit folgender Häufigkeitsverteilung gefunden:

Clusteranzahl	Seed	Ergebnis
8	2	keine sinnvollen Ergebnisse
8	2,5	keine sinnvollen Ergebnisse
8	7,43	keine sinnvollen Ergebnisse
8	100	keine sinnvollen Ergebnisse
8	200	keine sinnvollen Ergebnisse
8	1000	keine sinnvollen Ergebnisse
8	1500	keine sinnvollen Ergebnisse
8	-2	keine sinnvollen Ergebnisse
8	-10	keine sinnvollen Ergebnisse
8	-100	keine sinnvollen Ergebnisse
8	-12344	keine sinnvollen Ergebnisse
9	1	sinnvolle Ergebnisse (ein Ausreißer)
9	2	keine sinnvollen Ergebnisse
9	10	keine sinnvollen Ergebnisse
9	50	keine sinnvollen Ergebnisse
9	100	keine sinnvollen Ergebnisse
9	200	keine sinnvollen Ergebnisse
9	-1	sinnvolle Ergebnisse
9	-5	keine sinnvollen Ergebnisse
9	-25	keine sinnvollen Ergebnisse
9	-100	keine sinnvollen Ergebnisse
9	-200	keine sinnvollen Ergebnisse
9	-2345	keine sinnvollen Ergebnisse
10	2	keine sinnvollen Ergebnisse
10	5	keine sinnvollen Ergebnisse
10	50	keine sinnvollen Ergebnisse
10	100	keine sinnvollen Ergebnisse
10	200	keine sinnvollen Ergebnisse
10	1123	keine sinnvollen Ergebnisse
10	1234	keine sinnvollen Ergebnisse
10	-1	keine sinnvollen Ergebnisse
10	-5	keine sinnvollen Ergebnisse
10	-12	keine sinnvollen Ergebnisse
12	1	keine sinnvollen Ergebnisse
18	13	keine sinnvollen Ergebnisse
24	15	keine sinnvollen Ergebnisse
30	20	keine sinnvollen Ergebnisse
100	-1	keine sinnvollen Ergebnisse

Tabelle 16.18: Clustering nach Geburtsdatum und Geschlecht

SQL-Query	select *KundenID*,*Geburtsdatum*, Geschelcht from *TestKunden*
Clusteralgorithmus	SimpleKMeans
Seed	-5
Anzahl vorgegebener Cluster	8

Tabelle 16.19: Einstellungen finale Clusterings

Cluster	Centroid	Inhalt/Einträge
0	1962 / 0	23
1	1963 / 1	20
2	1977 / 1	25
3	1945 / 1	25
4	1989 / 1	24
5	1977 / 0	15
6	1943 / 0	28
7	1988 / 0	23

Tabelle 16.20: Ergebnis finales Clustering

- Cluster 0 23 männliche erwachsene Kunden um 40 Jahre.
- Cluster 1: 20 weibliche erwachsene Kunden um 40 Jahre.
- Cluster 2 25 männliche Frührentner oder ältere Kunden.
- Cluster 3 28 weibliche Frührentner oder ältere Kunden.
- Cluster 4 25 männliche Twens
- Cluster 5 15 weibliche Twens
- Cluster 6 24 männliche jugendliche Kunden
- Cluster 7 23 weibliche jugendliche Kunden

Die Verteilung der Kunden in die 8 Cluster entspricht der Verteilung, die in den Kundendaten manuell ausgelesen werden kann. Es sind die vier Geburtsjahresschwerpunkte 1944, 1963, 1977 und 1988 gefunden worden. Zu jedem Schwerpunktjahr sind dabei sowohl männliche als auch weibliche Kunden identifiziert worden. Die Ergebnisse

des finalen Clusterings sind daher als zufriedenstellend zu bezeichnen, weshalb im Folgenden nach einer weiteren Attributkombination geclustert werden kann.

Clustering nach Geburtsdatum und PLZ

Das Ziel dieser Clusteranaylse ist es gewesen, Kundengruppen im Bezug auf ihr Geburtsdatum und die Postleitzahl ihres Wohnortes zu identifizieren. Die Parameterübergabe ist dabei so gestaltet worden, dass das Attribut *KundenID* vom Clusteralgorithmus ignoriert worden ist, wobei es jedoch bei der Ergebnisausgabe wieder berücksichtigt wird, um eine Zuordnung der Kunden zu den jeweiligen Clustern zu gewährleisten. Es sind verschiedene Parametereinstellungen getestet worden, wobei wiederum acht Cluster ausgegeben werden sollten. Diese Cluster sollen letztendlich eine Interpretation entsprechend der folgenden Aufzählung ermöglichen:

- 1. 25 erwachsene Kunden um 40 Jahre mit einer PLZ um 23123.
- 2. Frührentner oder ältere Kunden mit einer PLZ um 66666.
- 3. erwachsene Kunden um 40 Jahre mit einer PLZ um 66666.
- 4. jugendliche Kunden mit einer PLZ um 66666.
- 5. jugendliche Kunden mit einer PLZ um 23123.
- 6. Frührentner oder ältere Kunden mit einer PLZ um 23123.
- 7. Twens mit einer PLZ um 23123.
- 8. Twens mit einer PLZ um 66666.

1. Clustering Bei der manuellen Durchsicht der synthetisch erzeugten Testdaten sind, wie weiter oben bereits beschrieben, vier Jahresschwerpunkte entdeckt worden. In Verbindung mit den Postleitzahlen der Wohnorte der Kunden, die hauptsächlich aus zwei fiktiven Einzugsbereichen kommen, sollten bezüglich der Interpretierbarkeit wiederum acht Cluster gebildet werden. Es sind bei dem ersten Clustering die Parametereinstellungen aus Tabelle 16.21 zur Anwendung gekommen.

SQL-Query	select *KundenID*,*Geburtsdatum*, *PLZ* from *TestKunden*
Clusteralgorithmus	SimpleKMeans
Seed	1
Anzahl vorgegebener Cluster	8

Tabelle 16.21: Einstellungen 1. Clustering

Cluster	Centroid	Inhalt/Einträge
0	1939 / 23123	4
1	1978 / 23123	25
2	1989 / 23123	24
3	1984 / 66666	36
4	1945 / 23123	30
5	1964 / 23123	25
6	1953 / 66666	39

Tabelle 16.22: Ergebnis 1. Clustering

Ergebnis des ersten Clusterings Bei der Betrachtung des ersten Clusterings (vgl. Tabelle 16.22) sind Cluster gefunden worden, die eine sinnvolle Interpretation nicht zulassen. Es sind nicht die bei den Parametereinstellungen eingegebenen acht Cluster identifiziert worden. Bei den sieben gefundenen Clustern sind dabei zudem fünf Kundengruppen identifiziert worden, die aus dem Einzugsgebiet des Ortes mit der Postleitzahl 66666 kommen. Aufgrund der nicht zufriedenstellenden Ergebnisse des ersten Clusterings sind daher weitere Clusterings mit anderen Parametereinstellungen durchgeführt worden.

2. Clustering Aufgrund der nicht zufriedenstellenden Ergebnisse des ersten Clusterings sind die in Tabelle 16.23 abgebildeten Parametereinstellungen zur Anwendung gekommen.

Ergebnis des zweiten Clusterings Die in Tabelle 16.24 dargestellten Ergebnisse, geben die existierenden Kundengruppen in den synthetisch erstellten Testdaten bezüglich der gewünschten Interpretierbarkeit besser wieder als die Cluster der ersten Clusterings. Es sind acht

SQL-Query	select *KundenID*, *Geburtsdatum*, *PLZ* from *Test_Kunden*
Clusteralgorithmus	SimpleKMeans
Seed	-5
Anzahl vorgegebener Cluster	8

Tabelle 16.23: Einstellungen 2. Clustering

Cluster	Centroid	Inhalt/Einträge
0	1944 / 66666	20
1	1963 / 66666	21
2	1966 / 23123	18
3	1958 / 23123	9
4	1985 / 66666	34
5	1978 / 23123	25
6	1943 / 23123	32
7	1989 / 23123	24

Tabelle 16.24: Ergebnis 2. Clustering

Cluster identifiziert worden, wobei lediglich die Kunden aus Cluster 3 nicht entsprechend der angestrebten Kundengruppen zugeordnet worden sind. Anstatt der erwachsenen Kunden um 40 aus dem Postleitzahlenbereich um 23123 hätten hier Twens aus dem Postleitzahlenbereich 66666 identifiziert werden müssen.

Weitere Clusterings In der nächsten Phase des Clusterings ist wiederum mit einer weiten Streuung des Seed-Parameters (-1000 bis 10000) gearbeitet worden, wobei die Clusteranzahl in fast allen Fällen bei 8,9 oder 10 gelegen hat. Bei den zahlreichen Clusterings ist jedoch generell das Problem aufgetreten, dass die identifizierten Kundengruppen bezüglich ihrer Geburtsjahre nicht exakt den Kundengruppen entsprechen, die bezüglich der Interpretation angestrebt worden sind. Zudem sind häufig weiblich oder männliche Kundengruppen aus einem Geburtsjahresschwerpunkt nicht einem eigenen Cluster zugeordnet worden. Die Tabelle 16.25 enthält dabei alle weiteren durchgeführten Clusterings. Die Bezeichnung von sinnvollen bzw. nicht sinnvollen Ergebnissen entspricht dabei genauso wie die Behandlung

von Ausreißern den Ausführungen in Abschnitt 16.1.5.

finales Clustering Nach den vielfältigen Clusterings mit verschiedenen Parametereinstellungen unter WEKA ist letztendlich ein Clustering mit den Einstellungen aus Tabelle 16.26 durchgeführt worden.

Ergebnis des finalen Clustering Die in Tabelle 16.27 dargestellten Ergebnisse geben die existierenden Kundengruppen entsprechend der gewünschten Interpretierbarkeit gut wieder. Der Clusteralgorithmus hat dabei jeweils vier Kundengruppen aus dem Einzugsgebiet der beiden Postleitzahlenschwerpunkte 23123 und 66666 identifiziert. Die dabei gefundenen Centroide bezüglich der Geburtsjahre der jeweiligen Kunden geben die Verteilung der Daten aus den synthetisch angelegten Testdaten ebenfalls entsprechend wieder. Folgende Kundengruppen sind also bei dem finalen Clustering identifiziert worden, wobei sie exakt den Kundengruppen entsprechen, die bezüglich der leichten Interpretierbarkeit angestrebt worden sind:

Clusteranzahl	Seed	Ergebnis
8	0	keine sinnvollen Ergebnisse
8	1	keine sinnvollen Ergebnisse
8	2	keine sinnvollen Ergebnisse
8	5	keine sinnvollen Ergebnisse
8	10	keine sinnvollen Ergebnisse
8	13	keine sinnvollen Ergebnisse
8	50	keine sinnvollen Ergebnisse
8	100	keine sinnvollen Ergebnisse
8	999999	keine sinnvollen Ergebnisse
8	-1	keine sinnvollen Ergebnisse
8	-2	keine sinnvollen Ergebnisse
8	-5	keine sinnvollen Ergebnisse
8	-10	keine sinnvollen Ergebnisse
8	-100	keine sinnvollen Ergebnisse
8	-500	keine sinnvollen Ergebnisse
9	1	keine sinnvolle Ergebnisse
9	2	keine sinnvollen Ergebnisse
9	6	sinnvollen Ergebnisse (ein Ausreißer)
9	100	sinnvolle Ergebnisse (ein Ausreißer)
9	200	keine sinnvollen Ergebnisse
9	-1	sinnvolle Ergebnisse (ein Ausreißer)
9	-5	keine sinnvollen Ergebnisse
9	-10	keine sinnvollen Ergebnisse
9	-1000	keine sinnvollen Ergebnisse
10	0	keine sinnvollen Ergebnisse
10	1	keine sinnvollen Ergebnisse
10	2	keine sinnvollen Ergebnisse
10	50	keine sinnvollen Ergebnisse
10	100	keine sinnvollen Ergebnisse
10	200	keine sinnvollen Ergebnisse
10	-1	sinnvolle Ergebnisse (ein Ausreißer)
10	-1	keine sinnvollen Ergebnisse
10	-10	sinnvolle Ergebnisse (ein Ausreißer)
11	-1	keine sinnvollen Ergebnisse
11	-10	keine sinnvollen Ergebnisse
15	0	keine sinnvollen Ergebnisse
22	-1	keine sinnvollen Ergebnisse
100	-1	keine sinnvollen Ergebnisse

Tabelle 16.25: Clustering nach Geburtsdatum und PLZ

SQL-Query	select *KundenID*,*Geburtsdatum*, *PLZ* from *Test_Kunden*
Clusteralgorithmus	SimpleKMeans
Seed	3
Anzahl vorgegebener Cluster	9

Tabelle 16.26: Einstellungen finales Clustering

Cluster	Centroid	Inhalt/Einträge
0	1964 / 23123	25
1	1942 / 66666	16
2	1954 / 66666	10
3	1988 / 66666	25
4	1989 / 23123	24
5	1944 / 23123	34
6	1978 / 23123	25
7	1970 / 66666	24

Tabelle 16.27: Ergebnis finales Clustering

- Cluster 0: 25 erwachsene Kunden um 40 Jahre mit einer PLZ um 23123.
- Cluster 1: 16 Frührentner oder ältere Kunden mit einer PLZ um 66666.
- Cluster 2: 10 erwachsene Kunden um 40 Jahre mit einer PLZ um 66666.
- Cluster 3: 25 jugendliche Kunden mit einer PLZ um 66666.
- Cluster 4: 24 jugendliche Kunden mit einer PLZ um 23123.
- Cluster 5: 34 Frührentner oder ältere Kunden mit einer PLZ um 23123.
- Cluster 6: 25 Twens mit einer PLZ um 23123.
- Cluster 7: 24 Twens mit einer PLZ um 66666.

SQL-Query	select *KundenID*,*Geburtsdatum*,*PLZ*, *Geschlecht* from *Test_Kunden*
Clusteralgorithmus	SimpleKMeans
Seed	3
Anzahl vorgegebener Cluster	19

Tabelle 16.28: Einstellungen 1. Clustering

Clustering nach PLZ, Geburtsdatum und Geschlecht

Die bisher gefundenen Kundengruppen setzen sich jeweils aus drei Attributen zusammen. Daher bezieht sich dieses Clustering auf vier Kundenattribute, wobei bspw. weibliche, jugendliche Kunden aus einem bestimmten Einzugsgebiet identifiziert werden sollen. Bei dem Clustering ist das Attribut *KundenID* wiederum ignoriert worden, wobei es aber bei der Ergebnisausgabe ausgegeben wird, weil es wichtig für die Zuordnung zu den Kundengruppen ist. Da dieses Clustering auf vier Attributen auf Grund von Zeitproblemen nicht notwendig gewesen ist, sind bei diesem Clustering weniger Parametereinstellungen getestet worden. Es ist jedoch bei diesen Versuchen auch relativ schnell klar geworden, dass es kaum möglich sein wird, 16 Cluster zu identifizieren, was bezüglich der Interpretierbarkeit jedoch als am sinnvollsten erscheint. Die Anzahl der 16 zu findenden Cluster resultiert dabei aus zwei Geschlechtern, zwei Einzugsgebieten der Kunden und den vier Geburtsjahresschwerpunkten der Kunden innerhalb der Testdaten.

1. Clustering Das erste Clustering auf den vier beschriebenen Attributen ist mit den Einstellungen aus Tabelle 16.28 durchgeführt worden.

Ergebnis des ersten Clusterings Die Resultate des ersten Clusterings, angezeigt in Ergebnistabelle 16.29, geben die Verteilung der Kundenattribute in den Testdaten nicht entsprechend der gewünschten Form wieder. Besonders deutlich wird dieses bei der Kundengruppe weibliche/männliche Twens. In den Clustern 7, 11, 12 und 14 werden diese alle dem Einzugsgebiet um die Postleitzahl 23123 zugeordnet. Diese entsprechen allerdings nicht den Clustern, die bezüglich

Cluster	Centroid	Inhalt/Einträge
0	1944 / 23123 / 0	14
1	1944 / 66666 / 0	15
2	1963 / 66666 / 0	13
3	1989 / 66666 / 1	13
4	1990 / 23123 / 1	12
5	1990 / 23123 / 0	11
6	1962 / 23123 / 1	13
7	1976 / 23123 / 1	13
8	1971 / 66666 / 1	13
9	1986 / 66666 / 0	15
10	1948 / 66666 / 1	6
11	1984 / 23123 / 0	1
12	1978 / 23123 / 0	9
13	1944 / 23123 / 1	19
14	1980 / 23123 / 1	5
15	1964 / 66666 / 0	11

Tabelle 16.29: Ergebnis 1. Clustering

ihrer Interpretierbarkeit als sinnvoll zu bezeichnen sind. Ansonsten ist dieses Clustering als zufriedenstellend anzusehen. Bei weiteren Clusterings mit variierten Parametereinstellungen unter WEKA sind in der Folgezeit keine besseren Ergebnisse erzielt worden. Innerhalb der Projektgruppe beschränkt sich das Finden von Kundengruppen mittels Clusterings daher ausschließlich auf die weiter oben erläuterten Clusterings, bei denen jeweils lediglich drei Attribute zur Identifikation von Kundengruppen zur Anwendung gekommen sind.

16.1.6 Clusterings auf der Händler 1-Datenbank

In diesem Schritt sind die synthetisch erstellten Testdaten, die sich bisher noch auf der Kartenanbieterdatenbank befunden haben, in die Datenbank von Händler 1 (siehe Zwischenbericht [iH03])) integriert worden. Es mussten dabei die bisher erstellten 1005 Kundendaten von Händler 1 beibehalten werden, da innerhalb der Datenbank Beziehungen zu anderen Relationen, wie zum Beispiel den gekauften Produkten in den Warenkörben, bestehen. Die 183 synthetisch erstellten Kundendaten sind daher komplett verfünffacht worden. Um jedoch die 1005 Kundendaten aus der Händler 1-Datenbank zu vervollständigen, mussten die ersten neunzig (1005 - 5*183) synthetisch erstellten Testdaten nicht verfünffacht- sondern versechsfacht werden. Auf diesen 1005 Kundendaten aus der Händler 1-Datenbank ist nun wiederum der SimpleKMeans-Algorithmus mit den weiter oben erprobten Parametereinstellungen angewendet worden.

Clustering nach Geschlecht und Geburtsjahr

Aufgrund der nicht gleichmäßig durchgeführten Vervielfachung der synthetisch angelegten Testdaten, haben die vorher erprobten Parametereinstellungen (Clusteranzahl 8, Seed -5) Kundengruppen ergeben, die im Bezug auf die angesprochene Interpretierbarkeit der Cluster keine entsprechenden Ergebnisse geliefert haben. Von daher mussten die Parametereinstellungen, wie in Tabelle 16.30 zu sehen, etwas korrigiert werden.

Ergebnis des Clusterings nach Geschlecht und Geburtsjahr Die geringe Abweichung in der Verteilung der Kundendaten hat dazu geführt, dass andere Parametereinstellungen zur Anwendung kommen mussten. Es sind dabei jedoch, wie in den Tabellen 16.31 und

SQL-Query	select knd-id,gebdatum, geschlecht from *Kunden*
Clusteralgorithmus	SimpleKMeans
Seed	-30
Anzahl vorgegebener Cluster	10

Tabelle 16.30: Einstellungen knd-id, gebdatum, geschlecht

Cluster	Centroid	Inhalt/Einträge
0	1978 / 0	81
1	1943 / 0	157
2	1964 / 1	105
3	1978 / 1	134
4	1990 / 1	140
5	1945 / 1	137
6	1989 / 0	127
7	1964 / 0	124

Tabelle 16.31: Ergebnis knd-id, gebdatum, geschlecht

16.20 zu sehen ist, fast identische Kundengruppen gefunden worden. Diese lassen sich nun für die Personalisierung innerhalb des Shops verwenden (vgl. Abschnitt 16.1.7).

Clustering nach PLZ und Geburtsjahr

Aufgrund der nicht gleichmäßig durchgeführten Vervielfachung der synthetisch angelegten Testdaten haben auch hier die vorher erprobten Parametereinstellungen (Clusteranzahl 9, Seed 3) keine sinnvoll zu interpretierenden Kundengruppen ergeben. Daher mussten die Parametereinstellungen, wie in Tabelle 16.32 zu sehen, wiederum etwas korrigiert werden.

Ergebnis des Clusterings nach PLZ und Geburtsjahr Die Abweichung in der Verteilung der Kundendaten hat dazu geführt, dass die Parametereinstellungen variiert werden mussten. Es sind dabei jedoch im Vergleich zum Clustering auf den synthetisch erzeugten Testdaten, wiederum fast identische Kundengruppen gefunden worden (vgl.

SQL-Query	select to char(k.gebdatum, 'YYYY') ,a.plz from *Adresse* a, *Kunde* k, *Adresse-Kunde* ak where k.knd-id=ak.knd-id and k.knd-sid=ak.knd-sid and ak.a-id=a.a-id and ak.a-sid=a.a-sid and ak.typ=1
Clusteralgorithmus	SimpleKMeans
Seed	5
Anzahl vorgegebener Cluster	9

Tabelle 16.32: Einstellungen knd-id, gebdatum, plz

Cluster	Centroid	Inhalt/Einträge
0	1965 / 23123	121
1	1944 / 23123	135
2	1977 / 66666	83
3	1989 / 66666	145
4	1944 / 66666	150
5	1989 / 23123	152
6	1963 / 66666	78
7	1978 / 23123	140

Tabelle 16.33: Ergebnis knd-id, gebdatum, plz

Tabelle 16.33 und Tabelle 16.27). Diese gefundenen Kundengruppen lassen sich ebenfalls für die Personalisierung innerhalb des Shops verwenden (vgl. Abschnitt 16.1.7).

16.1.7 Anwendung im Online-Shop

Das vorgestellte Clustering zur Identifikation von Kundengruppen kommt im Projektgruppenszenario im Shop zum Einsatz. Das Ziel ist dabei, den Kunden ein personalisiertes Angebot zu unterbreiten. Dabei kommen die beim Clustering entdeckten Kundengruppen, wie im folgendem beschrieben, zur Anwendung. Die jeweilige ID, die zur Identifikation der Kunden notwendig ist, wird zur Zuordnung der einzelnen Kunden zu den entdeckten Kundengruppen verwendet. Meldet sich ein Kunde nun mit seiner ID in den Online-Shop ein, so werden ihm Produkte angeboten, die die Kunden des gleichen Clusters vorher im Online-Shop gekauft haben. Die Zuordnung des jeweiligen Kunden zu einem Cluster kann aus der Relation *CLUSTKUNDENGEBGES* bzw. *CLUSTKUNDENGEBPLZ* bestimmt werden. Die Produkte der jeweiligen Cluster können anschließend aus der Relation *PRODUKTEGEBGES* bzw. *PRODUKTEGEBPLZ* gewonnen werden. Um den Personalisierungsaspekt dabei noch zu verstärken, ist der Online-Shop auf eine Anzeige der TOP5-Produkte beschränkt. Der angemeldete Kunde wird also einem Cluster zugeordnet, wobei ihm die fünf Produkte angeboten werden, die Kunden des gleichen Clusters bisher am häufigsten gekauft haben. So ist es zum Beispiel denkbar, dass Twens aus Oldenburg an erster Stelle der TOP5-Angebote Inliner angeboten werden, weil viele von ihnen an den Oldenburger Inliner-Nächten teilnehmen. An zweiter Stelle der TOP5 könnten zum Beispiel Pullover mit dem Aufdruck Oldenburg angeboten werden, die im letzten Sommer als Trend in dieser Zielgruppe gegolten haben. Auf diese Weise werden den Kunden des Online-Shops, für seine Kundengruppe und daher auch für den Kunden selbst, personalisierte Angebote unterbreitet.

16.1.8 Technisches Handbuch

Bei dieser Analyse der Kunden nach den Attributen *Geburtsdatum* und *Geschlecht* wird der *AnalysisRunner* aus dem Paket *diko.framework* mit der Konfigurationsdatei *anarunnergebplz.cfg* aufgerufen (siehe Anhang). Neben dem Aufruf des SimpleKMeans-Algorithmus mit

der Datei *ConfigKIDGebGes* (siehe Anhang) als Parameter wird der Postprozessor *diko.Personalisierung.kundenattribute.PostprocessorGeb-Plz* gestartet, der für die gefundenen Kundencluster jeweils die fünf meistgekauften Produkte bestimmt. Die Ausgangsdaten bilden die Clusterergebnisse in der Tabelle *ClustKundenGebGes* der Datenbank des Onlinehändlers und die Resultate dieses Nachbereitungsschrittes werden in der Tabelle *ProdukteGebGes* gespeichert. Die Funktionalität des Postprozessors ist dabei unabhängig von der Anzahl der gefundenen Cluster und auch von der Anzahl der Kunden in den einzelnen Clustern.
Analog verläuft die Analyse mit den Attributen Geburtsdatum und Postleitzahl, die notwendigen Dateien bzw. Tabellen sind:

- *anarunnergebplz.cfg*
- *ConfigKidGebPlz*
- *ClustKundenGebPlz*
- *ProdukteGebPlz*

16.2 Clusteranalyse nach Kunden und gekauften Produkten

Im folgenden wird die Umsetzung, der im Konzept vorgestellten Clusteranalyse nach Kunden und gekauften Produkten, beschrieben.

16.2.1 Aufgabenstellung und Zielsetzung

Zu der im Konzept vorgeschlagenen Analyse, ein Clustering der Kunden nach ihren gekauften Produkten, soll hier nun Umsetzungbeschrieben werden. In dieser Analyse werden die erworbenen Artikel eines Kunden nach den vorhandenen Warengruppen aufsummiert und der Anteil der einzelnen Warengruppen an der Gesamtmenge gespeichert (siehe 15.1.2). Aufgrund dieser relativen Werte sollen durch Clusteranalyse Kunden mit ähnlichem Profil zusammengefasst werden. Als Datengrundlage dient die in der Kartenanbieterdatenbank (siehe Zwischenbericht [iH03]) integrierten Daten des Onlinehändler, welche in der früheren Phase des Datenbankentwurfes angelegt worden sind.

Das Ziel dieses Arbeitsabschnittes ist die Generierung von Analyseergebnissen auf der Basis der gegebenen Händlerdaten und eine Betrachtung einer möglichen Umsetzung der Resultate in Form einer Personalisierung des zu Evaluierung angelegten Online-Shops. Weiterhin sollen durch die Arbeit mit den vorhandenen Ressourcen Hinweise für eventuelle Verbesserung bzw. neue Wege bei der Analyse gewonnen werden.
Um die geforderten Ziele zu erreichen, werden am Anfang aus den Daten des Onlinehändlers die notwendigen Daten ausgewählt, bearbeitet und in einem für die Analyse geeignetem Format gespeichert. Für diese Vorverarbeitungsschritte wird ein eigenes Java-Paket bereitgestellt. Die Analysen werden je nach benötigter Information und gewählten Verfahren mit Hilfe des Framework von Diko (siehe 14) oder der WEKA-GUI [WF01] durchgeführt. Die Ergebnisse der Analyse sind anschließend auszuwerten, um Rückschlüsse für folgende Analyseschritte gewinnen zu können. Der experimentelle Charakter der Aufgabe verlangt einen stetigen Wechsel zwischen den Phasen der Vorverarbeitung, Analyse und Bewertung der Ergebnisse, wobei die Anwendbarkeit der Resultate im Online-Shop als Abbruchkriterium festgesetzt ist.
In Abschnitt 16.2.2 wird die Auswahl der Daten als auch deren Aufbereitung für die beabsichtigten Analysen beschrieben. Kapitel 16.2.3 beschäftigt sich mit der Durchführung der Analyse und der Auswertung ihrer Ergebnisse. Die Verwendung der Ergebnisse steht hingegen im Mittelpunkt des Kapitels 16.2.4; das Fazit (Abschnitt 16.2.5) schließt den Text ab, der Anhang beinhaltet die genauen Ergebnisse der durchgeführten Analysen.

16.2.2 Auswahl der Daten und Vorverarbeitung

Um die angestrebten Analysen mit Hilfe von WEKA durchführen zu können, müssen die relevanten Daten selektiert werden (Abschnitt 16.2.2), neue Werte berechnet und diese abschließend in ein geeignetes Format gespeichert werden (Abschnitt 16.2.2).

Datenauswahl

Für die geforderte Analyse sind die gekauften Artikel eines jeden Kunden zu bestimmen und diese werden dann ihrer jeweiligen Warengruppe zugeordnet. Die Artikel einer Warengruppe werden anschließend

aufsummiert und der Anteil dieser Warengruppe an der Gesamtmenge der Artikel des jeweiligen Kunden berechnet.
Die benötigten Information befinden sich in der Kartenanbieterdatenbank in den Tabellen *Kunden, Transaktion, Posten und Einordnung.* Aus der Relation Kunden sind die IDs der Kunden zu bestimmen, welche in der Tabelle *Transaktion* im Attribut *Inhaber* aufgeführt sind. Die einzelnen Produkte und deren Anzahl sind in der Tabelle *Posten* gespeichert. Die Verknüpfung der Produkte mit einer Warengruppe ist wiederum in der Relation *Einordnung* gegeben. Diese bilden neben den für die Verknüpfungen benötigten Fremdschlüsseln die Menge der relevanten Daten für die angestrebte Analyse.

Aufbereiten der Daten

Um den Aufwand für die notwendigen SQL-Anfragen zu minimieren, wurde eine sogenannte MaterializedView *gana*, eine Art Zwischentabelle, in dem Schema der Kartenanbieterdatenbank erstellt, um nicht für jeden Kunden aufwendige Tabellenverknüpfungen (Joins) durchführen zu müssen. Die SQL-Anweisung zum Erzeugen dieser Sicht lautet:

```
create materialized view gana as select
p.transaktionid, p.anzahl, e.warengruppeid
from einordnung e join posten p on
(p.produktid=e.produktid and p.tza_produkt=e.tza_produkt)
where p.haendlerid=1 and e.haendlerid=1
refresh complete
```

Dieser Schritt war notwendig, da die zu verknüpfenden Tabellen Einordnung und Posten eine bzw. zehn Millionen Einträge besitzen. Aus der Relation *Posten* werden die Werte der Attribute *Anzahl* und *Transaktionid* und aus der Relation *Einordnung* die WarengruppeID der jeweiligen Produktes eines Postens ermittelt. Hier wie auch bei allen anderen Anfragen wird die Auswahl auf die Daten des Onlinehändlers (Attribut HändlerID=1) beschränkt. Als nächster Schritt werden die Kundennummern (Attribut *KundenID*) aller Kunden des Onlinehändlers aus der Tabelle *Kunden* ausgelesen. Für jeden Kunden werden nacheinander zuerst die getätigten Transaktionen aus der Tabelle Transaktion ausgelesen, um anschließend die mit den Transaktionen verbundenen Einträge in der View *gana* zu ermitteln. Da-

bei wird bei jedem gefundenen Eintrag der Zähler der entsprechenden Warengruppe *(gana.warengruppeid)* um die jeweilige Anzahl (*gana.anzahl*) erhöht. Für jeden Kunden wird abschließend ein Eintrag in die Tabelle *ClusteringKngP* der Kartenanbieterdatenbank mit dem prozentualen Werten für die zur Zeit sieben Warengruppen (Attribute WG1 bis WG7) vorgenommen. Der hier beschriebene Prozess ist in dem Java-Paket *diko.personalisierung.personalisierung* realisiert worden. Vor einer Analyse ist die Klasse *Personalisierung* auszuführen, um die aktuellen Daten in die Tabelle *ClusteringKngP* zu transferieren. Bei Erweiterung der Warengruppenhierarchie sind diese auch in den entsprechenden Klassen des Paketes durchzuführen. Weiterhin werden die Join-Operation durch Indizes auf den Primär- bzw. Fremdschlüsseln der verwendeten Tabellen unterstützt. Auch diese sind bei stark gewachsener Datenmenge auf einen aktuellen Stand zu setzen.

Notwendige Schritte bei Änderung der Datenbasis

Sollte die Datenbasis der Analyse geändert oder erweitert werden so sind einige Anpassungen des Verfahrens vorzunehmen. Die materialisierte Sicht *gana* ist durch den Befehl immer aktuell und muß daher nicht behandelt werden. Ändert sich die Anzahl der Warengruppen, so muss dieses in der Klasse *diko.personalisierung.personalisierung.ClusteringKngP* angepasst werden, indem wie für die bisherigen sieben Gruppen ein Attribut angelegt und dieser bei der Berechnungen (Gesamtposten und Anteil) mit berücksichtigt wird. Die Tabelle *ClusteringKngP* in der Kartenanbieterdatenbank ist ebenfalls anzupassen (Löschen bzw. Hinzufügen von Attributen).

16.2.3 Analyse und Auswertung

Die Analysen basieren auf der neu erzeugten Tabelle ClusteringKngP in der Kartenanbieterdatenbank. Die Anweisung zum Auslesen der notwendigen Daten lautet:

```
SELECT * FROM CLUSTERINGKNGP
```

Die Analysen wurden mit der GUI von WEKA durchgeführt, wobei unterschiedliche Algorithmen angeboten werden (SimpleKMeans, cobbweb und EM). Der Algorithmus Cobweb erwies sich dabei als

nicht geeignet für diese Analyse, da für eine weitere Verarbeitung nicht nur die Cluster sondern auch deren Centroide von Interesse sind, diese aber nicht mit angegeben werden. Der zweite Algorithmus EM liefert zwar die Information der Mittelpunkte, doch die Unterschiede der Centroide der Cluster sind so minimal, dass die Unterteilung der Kunden in die gefundenen Gruppen als nicht sehr sinnvoll erscheint. Die Veränderungen der Parameter haben dabei keinen großen Einfluss auf diese Erscheinung. Die Ergebnisse einiger Analysen sind dem Anhang zu entnehmen (Abschnitt VI).

Der dritte Algorithmus Simple KMeans erwies sich als beste Wahl unter den drei zur Verfügung gestellte Verfahren, weswegen dieser hauptsächlich zum Test der Analyse verwendet wurde. Die Ergebnisse dieser Analysen sind ebenfalls dem Anhang zu entnehmen (VI). Der Algorithmus ist über die beiden Parameter seed, der einen Initialwert für die Berechnungen vorgibt und die maximale Anzahl an Clustern zu steuern. Es wurde willkürlich der Wert 5 für die Clusteranzahl zu Beginn gewählt und mit verschiedenen seed-Werten kombiniert. Auffällig war hierbei, dass bei einem seed von -1 nicht die Maximalzahl an Clustern genutzt wurde, sondern die Käufer nur in zwei große Gruppen unterschieden wurden. Die Differenzierung erfolgt hauptsächlich beim dem Attribut *WG1* (12 zu 16 Prozent[1]). Da bei den meistens Analysen die Maximalanzahl erreicht wurde, wird die Clusteranzahl auf 8 erhöht und eine ähnliche Testreihe durchgeführt (seed=(-5, -4, -3, -2, -1, 0, 1, 2, 4, 5, 8, 20)). Die erste interessante Beobachtung ist, dass bei seed=-4 das gleiche Resultat wie bei fünf Clustern und seed=-1 entsteht (siehe oben). Bei einem seed von größer 4 wird nicht mehr die Maximalzahl erreicht, da mehrere Cluster den Centroiden (0,0,0,0,0,0,0) besitzen und die Nichtkäufer nur einem Cluster zugeordnet werden. Eine weitere Erhöhung der Clusteranzahl auf 12 ergibt die Erkenntnis, dass sich die Kunden auf die Mehrzahl der Cluster verteilen und somit kleinere Gruppen entstehen. Der nächste Schritt auf 20 Cluster ergibt für die Centroiden teilweise Werte von unter 10 bzw. über 22 Prozent, die bisher in den Analysen nicht aufgetreten sind. Das Intervall der Einträge (5-28 Prozent) in der zu Grunde liegenden Tabelle wird also weiter ausgereizt, das Gros der Werte liegt aber weiterhin bei 13 bis 17 Prozent.

Um die beobachteten Entwicklungen weiter zu untersuchen, wurden

[1]Die Angaben in Prozent beschreibt das Verhältnis der gekauften Artikel einer Warengruppe zu der Gesamtmenge der gekauften Artikel

im letzten Schritt auch Analysen mit einer Maximalzahl von 100 bzw. 200 Clustern durchgeführt. Auch hier ist der Trend zu erkennen, dass immer mehr Extremwerte des Intervalls erreicht werden (7-23 Prozent) und zudem die viele Centroidenwerte sich aus dem Bereich von 13-17 Prozent zu den Rändern hin entfernen. Bei diesen Analysen entstehen außer den Nichtkäufern keine größeren Cluster, die mehr als 2 Prozent der Kunden enthalten.

Unabhängig vom gewählten Umfeld und Verfahren bleibt demnach festzuhalten, dass aus den gegebenen Daten keine Ergebnisse zu erlangen sind, die für weitere Aufgaben genutzt werden können. Um ein generelles Fehlverhalten der Algorithmen ausschließen zu können, sind eine überschaubare Mengen von Testdaten im gleichen Format angelegt worden, aus denen nachvollziehbare Cluster gebildet werden können. Aufgrund dieser Ergebnisse wurde auf eine Analyse mit Hilfe des entwickelten Frameworks verzichtet.

Bei näherer Betrachtung der Ausgangsdaten in der Relation ClusteringKngP wird die große Ähnlichkeit der Daten sichtbar: Die Werte der per tBasket erzeugten Daten (KundenID größer 4) haben für die Warengruppe Minimalwerte zwischen 5 und 7 Prozent und Maximalwerte zwischen 22 und 28 Prozent. Der größte Anteil der Werte bewegt sich aber in dem Intervall von 12 bis 18 Prozent, aus dem dann auch die meistens Werte für die Centroide der gefundenen Cluster rekrutieren. Auffällig an den Testdaten ist die große Gruppe der Nichtkäufer (324 Kunden), die auch bei jeglicher Analyse als eigenständiges Cluster bestimmt werden konnten. Eine bessere Arbeitsgrundlage wären reale Transaktionsdaten gewesen und nicht künstlich erzeugte Massendaten, die zwar zeitliche Muster beinhalten, ansonsten aber willkürlich durch Skripte erzeugt worden sind.

16.2.4 Verwendung der Analyseergebnisse

Im diesem Abschnitt werden nun die Möglichkeiten beschrieben, wie die im vorangegangenen Kapitel gefundenen Ergebnisse genutzt werden können. Dabei wird zwischen der konkreten Verwendung im Rahmen des Projektes und einer allgemeinen unterschieden, da aufgrund der gegebenen Daten nicht alle Potentiale der Analyse ausgeschöpft werden konnten.

Umsetzung im Rahmen der Projektgruppe

Die durch die Analyse erzeugten Cluster sind aufgrund ihrer Qualität weder für eine direkte Umsetzung noch als Grundlage für weiterführende Analyseverfahren wie eine Assoziationsanalyse zu verwerten. Das Cluster der Nichtkäufer bietet keine Aussagekraft für eine Personalisierung und die anderen beiden Hauptgruppen sind sich so ähnlich, das eine Differenzierung nicht sinnvoll erscheint. Weiterhin ist zu beobachten, dass die meisten Werte zwischen 12 und 17 Prozent liegen und somit der Verdacht einer Gleichverteilung der Einkäufe über die Warengruppen naheliegt. Auch dieser Effekt ist auf die Art der Datenerstellung zurückzuführen. Die für diese Analysen angedachte Verwertung der Ergebnisse wird daher im nächsten Abschnitt theoretisch erörtert.

Projektunabhängige Umsetzungsmöglichkeiten

Die Cluster der Analyse Kunden nach gekauften Produkten können nur in Zusammenhang mit den Centroiden der Cluster verwendet werden. Die Kunden werden zusammen mit der Nummer ihres Clusters in der Zieldatenbank gespeichert. Sofern ein Kunde im Online-Shop angemeldet ist, können sein Cluster bestimmt und über die Daten des Mittelpunktes die wichtigsten Kategorien als direkte Links auf der Startseite aufgeführt werden. Zur Bestimmung der Gruppen kann eine gewisse Anzahl oder eine prozentuale Grenze (z.B. alle über 20 Prozent) gewählt werden. Das genaue Verfahren ist aber stark von der Anzahl der Warengruppen abhängig. Im Rahmen des Projektes wurde dieser Aspekt nicht weiter verfolgt, da die Daten keine sinnvollen Tests zulassen.
Ein zweiter Weg ist die Verwendung der gefundenen Cluster als Grundlage für weitere Analysen: wie schon in dem Konzept erwähnt, wäre die Assoziationsanalyse eine mögliche Wahl. Die für das jeweilige Cluster gefundenen Regeln könnten für einen Kunden des Clusters, der den ersten Teil der Regel erfüllt hat, angewandt werden, in dem ihm die Folge dieser Regel als angeboten wird.

16.2.5 Fazit

Als Fazit dieser Analyse ist festzuhalten, dass die Generierung der Testdaten mit mehr Sorgfalt und vor allem mit Sicht auf die bevorstehenden Aufgaben betrieben werden sollte. Eine Verwendung

von realen Handelsdaten wäre eine Alternative, um Kundenprofile zu erhalten, mit denen die Personalisierung besser durchgeführt werden könnte. Jedoch standen der Projektgruppe Echtdaten nicht zur Verfügung.

16.3 Clusteranalyse von Warenkörben nach Produkttypen

Im Folgenden wird die Umsetzung des Konzeptes „Clustering von Kunden und gekauften Produkten“ (siehe Kapitel 15) erläutert. Wie unter Abschnitt 16.3.1 beschrieben, ist die Analyse auf Produktebene jedoch zu feingranular, um praktisch durchgeführt zu werden. Daher wird von den Produkten abstrahiert und es werden nur Produkttypen beachtet, die z.B. aus der Kategoriezugehörigkeit abgeleitet werden können. Das Ziel dieser Analyse soll sein, eine Menge von Warenkörben so einzuteilen, dass Cluster mit Warenkörben entstehen, die ähnliche Produkte enthalten.
Die Ergebnisse dieses Verfahrens können im Anschluss als Grundlage weiterer Analysen, im Speziellen von Assoziationsanalysen, verwendet werden. Damit ist diese Analyse ebenfalls Teil der Umsetzung des Konzeptes „Clustering nach Kunden und gekauften Produkten in Verbindung mit einer Assoziationsanalyse“ (siehe ebenfalls Kapitel 15). Die einzelnen Cluster enthalten voraussichtlich Warenkörbe, die bereits eine Ähnlichkeit aufweisen, also Produkttypen enthalten, die ohnehin oft zusammen gekauft werden. Die Assoziationsanalyse arbeitet somit auf einer Menge von Warenkörben, die bereits eine gewisse Ähnlichkeit haben. Wurde beispielsweise ein Cluster mit Warenkörben, in denen Chips und Bier enthalten sind, durch das Clustering gefunden, könnte die Assoziationsanalyse nun feststellen, dass Chio Chips oft mit Beck’s und Bahlsen Chips oft mit Jever zusammen verkauft werden. In dem Cluster kann der Support der Regeln relativ hoch sein, wobei die Regeln unter allen Warenkörben vielleicht einen geringen Support haben und somit nicht gefunden werden.

16.3.1 Umsetzung

Bei der Analyse der Warenkörbe ist zu beachten, dass Warenkörbe in der Regel in einer $1 : n$ Beziehung gespeichert werden. Jedem Warenkorb können demnach beliebig viele Produkte zugeordnet sein.

WEKA kann jedoch nur mit einer festen Anzahl von Attributen umgehen, so dass die Warenkörbe so transformiert werden müssen, dass jeder Warenkorb eine feste Anzahl von Attributen enthält.
Die sinnvollste Lösung dieses Problemes ist, nicht die Produkte, sondern die Produkttypen zu beachten. Diese können entweder aus dem Produktkatalog gewonnen werden, oder sind wie im Falle des Online-Händlers explizit vorhanden. Zusätzlich hat dieses Vorgehen den Vorteil, bessere Ergebnisse zu liefern. Dies geht aus der Tatsache hervor, dass beispielsweise oft Chips und Bier zusammen gekauft werden, aber die Marken durchaus variieren können. Werden Produkttypen betrachtet, wird auf einer höheren Abstraktionsebene gearbeitet. Details wie die Marke werden ignoriert und es wird nur auf einer Ebene wie Chips, Bier u.s.w. unterschieden.
Die Transformation der Daten wurde so implementiert, dass eine neue Tabelle (siehe Tabelle 16.34) erstellt wurde, die zu jedem Warenkorb speichert, ob ein Produkt eines bestimmten Typs im Warenkorb enthalten war oder nicht. Dabei wurden die Mengen der Produkte ignoriert, die Zugehörigkeit ist somit als binäres Attribut realisiert. Für die Personalisierung kann aus den Produktmengen ebenfalls keine weitere interessante Information gewonnen werden. Da durch die Analyse entweder Kategorien herausgefunden werden, die dem Benutzer vorgeschlagen werden können, oder Assoziationsregeln auf den Clustern angewendet werden. Im ersten Fall würde die vorgeschlagene Kategorie unabhängig von der Menge gleich bleiben und im zweiten Fall würden die Mengen die Assoziationsanalyse nicht beeinflussen, da diese an sich schon keine Mengen betrachtet.
Jedes Produkt hat des Weiteren genau einen Typ. Der Fall, dass ein Produkt zu mehreren Kategorien oder Typen gehört, wurde hier nicht speziell behandelt. In der Praxis kann dies zwar vorkommen, jedoch ist es wahrscheinlich, dass es auf einer Ebene passiert, die für dieses Analyseverfahren ohnehin zu feingranular wäre. Eine Implementierung dieser Analyse mit mehrfacher Typ-Zugehörigkeit pro Produkt wäre jedoch ohne Weiteres zu lösen, da einzig die Transformation dementsprechend angepasst werden müsste.
Die Attribute haben den Typ „nominal“ bekommen, da binäre Attribute nicht in WEKA existieren und keine reellen Zwischenwerte zugelassen werden sollen.
Als Algorithmus wurde XMeans [PM00] benutzt. Dieser Algorithmus hat den Vorteil, selber die optimale Clusteranzahl zu bestimmen. Es wurde eine eigene Distanzfunktion (*weka.core.NominalDistance*, siehe

Warenkorb-ID	Produkttyp 1	...	Produkttyp n
1	1	...	0
2	0	...	0
3	0	...	0

Tabelle 16.34: Schema der Zwischentabelle zum Warenkorb-Clustering nach Produkttypen

auch [Dik03]) für nominale Attribute implementiert, die bei Gleichheit des Wertes eine 0 und ansonsten 1 zurückliefert. Damit ist diese Distanzfunktion im Speziellen für die hier vorliegenden binären Attribute geeignet. Dennoch wäre es zu überlegen, eine bessere Distanzfunktion wie z.B. den Jaccard Koeffizienten [LWD98] zu verwenden, der die Gleichheit von Attributen höher bewertet.

16.3.2 Ergebnisse

Die Ergebnisse der Analysen fielen wie erwartet aus. Es wurden in den Testdaten des Kartenanbieter 18 Cluster gefunden. Die maximale Zahl von Clustern war 128. Da die Testdaten synthetisch erzeugt wurden, wäre es interessant, dieses Verfahren mit echten Daten zu testen. Bei kleineren Datenmengen, in denen bestimmte Cluster erzeugt wurden, konnte der Algorithmus genau die erzeugten Cluster wieder entdecken. Insofern kann davon ausgegangen werden, dass das Verfahren tatsächlich wie gewünscht funktioniert.
Im Folgenden werden kurz die verwendeten Parameter und die Auswirkungen durch Veränderung der Parameter aufgeführt. Dabei wird darauf verzichtet, alle benutzten Werte exakt aufzulisten, sondern es werden nur die Tendenzen aufgezeigt. Bei der Wahl der Parameter ist auch zu beachten, dass die Analysen im Online-Shop automatisch ausgeführt werden. Es ist also wichtiger Parameter zu finden, die allgemein gute Ergebnisse versprechen, als Parameter, die lediglich die Testdaten optimal clustern.

maxIterations Dieser Parameter gibt an, wie oft XMeans selber durchlaufen wird (Improve-Params, Improve-Structure). Die Testläufe wurden mit einem Wert von 5 durchgeführt. Allerdings wurden die 5 Durchläufe nie erreicht. Bei größeren Datenmengen können jedoch mehr Durchläufe nötig sein.

maxKMeans, maxKMeansStructure Setzt die maximale Anzahl der Wiederholungen im kMeans Algorithmus[Pre03] in den beiden XMeans Phasen. Die Ergebnisse waren gegenüber Veränderungen relativ stabil. Allerdings konnten durch größere Werte durchaus mehr und bessere Cluster gefunden werden. Werte von 10000 wurden als optimal angesehen, da größere Werte keine Verbesserungen mehr gebracht haben.

minNumClusters Gibt an, wieviele Cluster mindestens gefunden werden. Von dieser Clusterzahl aus wird versucht, durch das Splitten der Cluster bessere Centroide zu finden. Interessanterweise hatte dieser Parameter Auswirkungen. Bei Werten kleiner 5 hat er jeweils nur die minimale Zahl an Clustern gefunden. Ab 5 ging die Clusteranzahl plötzlich hoch. Es ist schwer zu sagen, warum sich der Algorithmus so verhält. Evtl. ist dies auf die Distanzfunktion oder auch auf das Bewertungsverfahren zurückzuführen.

maxNumClusters Gibt die maximale Anzahl Cluster an, die gefunden werden sollen. Wird dieser Wert erreicht, stoppt der Algorithmus. Aktuell wird dort ein Wert von 150 verwendet, der größer als die maximale Clusteranzahl ist und somit auf jeden Fall ausreichend gewählt ist.

cutOffFactor Bestimmt, wie viele der besten Splits verwendet werden, wenn die Bewertung ergeben hat, dass keins der Kinder-Centroide besser ist als der Vater-Centroid. Als optimal hat sich ein Wert von 0.5 erwiesen.

16.3.3 Anwendung der Ergebnisse im Shop

Es gibt zwei Möglichkeiten, die Ergebnisse im Shop zu verwenden. Die erste ist die dynamische Einblendung von Produktkategorien basierend auf der aktuellen Zusammensetzung des Warenkorbes. Zum Beispiel könnten jemandem, der bereits Produkte vom Typ Bier im Warenkorb hat, Kategorien vom Typ Chips vorgeschlagen werden. Der große Nachteil dieser, den Assoziationsregeln sehr ähnlichen, Methode ist, dass es sehr kompliziert ist, diese Zusammenhänge aus den Clustern zu erlangen. Es müssten alle Centroide mit dem aktuellen Warenkorb unter Zuhilfenahme der Distanzfunktion des Clusterers verglichen werden. Daraus kann eine Rangliste von Ähnlichkeiten

des aktuellen Warenkorbs zu den Centroiden generiert werden, indem Centroide mit einer geringeren Distanz als „besser“ und somit höher eingestuft werden. Anschließend wird der Centroid genommen, der in der Rangliste am höchsten steht und Produkttypen enthält, die noch nicht im aktuellen Warenkorb sind. Diese Produkttypen werden dem Nutzer dann vorgeschlagen. Da dies ein recht großer Berechnungsaufwand ist und ein Online-Shop auch gewissen Echtzeitanforderungen, wie vor allem einem kurzen Antwortzeitverhalten, gerecht werden muss, ist diese Methode an sich eher ungeeignet. Sie hat aber den Vorteil, weniger speziell zu sein als Assoziationsregeln. Während einem eine Assoziationsregel ein ganz bestimmtes Produkt vorschlägt, wird durch obige Methode nur eine Kategorie vorgeschlagen. Isst ein Kunde beispielsweise gerne Chips zu seinem Bier, mag aber keine Chips der Firma A, würde ihm eine Assoziationsregel, die ihm Chips vom Hersteller A vorschlägt, wahrscheinlich nicht zum Kauf animieren. Würde ihm allerdings der Vorschlag gemacht werden, Chips im allgemeinen zum Bier zu kaufen, könnte ihn das evtl. eher anregen.
Die zweite Anwendungsmöglichkeit der Analyseergebnisse besteht darin, die Ergebnis-Cluster als Grundlage einer Assoziationsanalyse zu nutzen. Ein großes Problem bei Assoziationsregeln ist es, Regeln zu finden, die einen ausreichenden Support haben, also in einer ausreichenden Anzahl der Warenkörbe gelten. Die Algorithmen arbeiten dazu mit einem minimalen Supportwert, den die Regel erfüllen muss, um als Regel erkannt zu werden. Wird dieser Supportwert zu hoch gewählt, um qualitativ hochwertige Regeln zu finden, besteht die Gefahr, gute Regeln zu verlieren, die vielleicht auf allgemein wenig gekauften Produkten gelten[2]. Das Clusteringverfahren soll genau das beschriebene Problem beheben. Es ist wahrscheinlich, dass die Warenkörbe mit dem Produkttyp A in einen eigenen Cluster eingeordnet werden. Wird die Assoziationsanalyse nur auf die Warenkörbe dieses Clusters angewandt, dürfte die oben erwähnte Regel einen deutlich höheren Support bekommen und würde somit erkannt werden, ohne dass der Supportwert zu niedrig gewählt werden muss. Ein zu niedriger Supportwert hat nämlich den Nachteil, Regeln zu fördern, die eher uninteressant sind.
Im Online-Shop würden diese Ergebnisse ebenso verwendet werden

[2]Der Supportwert wird immer auf alle Daten bezogen. Existiert also eine Regel, die zwischen den Produkten des Typs A einen hohen Support hat, aber treten Produkte dieses Typs nur in 5% der Warenkörbe auf, wäre der Support der Regel unter allen Warenkörben trotzdem sehr gering.

wie die einfachen Assoziationsregeln. Es werden also je nach Produkten im Warenkorb dynamisch Links zu assoziierten Produkten eingeblendet. Einzig die Analyse unterscheidet sich dadurch, dass vor der eigentlichen Analyse die Produkte aufgrund der Clusteringergebnisse gefiltert werden.

17 Konzept - Temporale Personalisierung

Das Ziel der temporalen Analysen ist es, zeitliche Zusammenhänge in den Transaktionsdaten festzustellen. Sind zeitliche Zusammenhänge bekannt, können temporale Personalisierungen vorgenommen werden. Dazu gehören beispielsweise Produkt-Bundles, die zu bestimmten Zeiten gültig sind oder aber auch saisonale Produkte, die in bestimmten Zeiträumen (z.B. Jahreszeiten) bevorzugt gekauft werden.
Die temporalen Analysen beschränken sich im Folgenden auf kalendarische Muster und temporales Clustering. Ein weiterer Ansatz, sogenannte „sequentielle Muster“, werden zwar am Ende kurz eingeführt, kann jedoch aus zeitlichen Gründen nicht umgesetzt werden.
Die temporalen Analysen wurden auf die vorhandenen Daten von Händler 1 beschränkt. Da das Framework an sich aber für verteilte Daten ausgelegt ist, sind ebenfalls Analysen auf anderen Daten möglich.

17.1 Kalendarische Muster

Für die Analyse wurde im ersten Abschnitt der Algorithmus zur Entdeckung kalendarischer Muster von Li et. al.[LNWJ01] ausgewählt. Die Projektgruppe erhofft sich von diesem Algorithmus Ergebnisse, die für die Personalisierung im Online-Shop verwendet werden können. Angestrebt ist das Vorschlagen ausgewählter Produkte aufgrund zeitlicher Assoziation zu einem anderen Produkt.
Im Folgenden wird die Funktionsweise allgemein beschrieben und dann die Implementierungsaspekte erläutert.

17.1.1 Ausgangsdaten

Die Analyse der kalendarischen Muster wird auf den Warenkorbdaten des Online-Händlers durchgeführt.

17.1.2 Analyseverfahren

Das Analyseverfahren nennt sich „Discovering Calendar-based Temporal Association Rules“; es handelt sich demnach um kalenderbasierte, temporale Assoziationsregeln. Muster, die in Anlehnung an ein *kalendarisches Schema*, z.B. einen Jahres-, Monats- oder Wochenkalender gesucht werden, nennt man kalendarische Muster. Ein kalendarisches Muster wird mit temporalen Assoziationsregeln beschrieben. Der temporale Aspekt einer solchen Regel könnte zum Beispiel „jeden Dienstag“, „jeden ersten Mai“ oder ähnlich lauten.
Um eine Regel aufstellen zu können, wird ein relationales Kalenderschema angelegt, dass z.B. aus Jahr, Monat und Tag besteht. Ein Muster ist dann ein Tupel in R mit der Form $\langle Jahr, Monat, Tag \rangle$. Wird das kalendarische Schema anders gestaltet, ergeben sich entsprechend andere Muster. So kann ein Schema auch aus z.B. Wochentag, Stunde und Minute bestehen.
Darüber hinaus gibt es die Möglichkeit „Wildcards“ zu verwenden. Wildcards sind Platzhalter, die für jeden beliebigen, aber zulässigen Wert stehen. Ein Wildcard wird mit einem Stern „$*$“ gekennzeichnet. Das Tupel $\langle *, 1, 10 \rangle$ steht dann für den 10. Tag im Januar in *jedem* Jahr. Eine temporale Assoziationsregel hat die Form (r, e) mit r als Assoziationsregel und e als kalendarischem Muster.

17.1.3 Interpretation

Die Ergebnisse des in [LNWJ01] beschriebenen Algorithmus werden wie folgt dargestellt:

(Produkt-ID 1322, Produkt-ID 2311 , $\langle *, 11, * \rangle$)

Das bedeutet, dass die Produkte 1322 und 2311 jeden Tag, in jedem Jahr im November zusammen gekauft werden. Grundsätzlich besteht ein Muster aus mindestens zwei Produkten. Es ist aber ebenso möglich, dass mehr als zwei Produkte zeitgleich (in diesem Kontext am selben Tag) zusammen gekauft werden. In diesem Fall wird sowohl das größtmögliche Muster dieser Produkte als auch alle „Untermuster“ als Ergebnis ausgegeben. Existiert z.B. ein Muster mit fünf Produkten, würde der Algorithmus sowohl dieses Muster als auch alle Teilmengen mit nur vier (drei und zwei) Produkten ausgeben.
Zu diskutieren ist, ob dieser Algorithmus für die angestrebte Personalisierung nützlich ist. Ein erster Kritikpunkt ist die Granularität,

da Regeln, die den Produktkauf zu einer bestimmten Tageszeit oder an einem bestimmten Wochentag beschreiben, nicht implementiert werden. Gerade die Unterstützung von Wochentagen (Montag, Dienstag, ...) ist schwierig, da diese einem anderen Schema folgen als Tage, Monate und Jahre. Doch speziell die Erweiterung auf ein Kalender-Schema, das Wochentage oder noch besser Tageszeiten (z.B. auf eine Stunde genau) angibt, wäre wünschenswert.
Wie ist der praktische Nutzen, wenn wir Muster mit der Genauigkeit eines Tages erhalten? Die Aussage, dass zum Beispiel an jedem ersten eines Monats, also $\langle 1, *, * \rangle$, Produkt 1 und Produkt 3 zusammen gekauft werden, könnte für die Personalisierung so verwendet werden, dass jeder Kunde, der eines der beiden Produkte an einem solchen Tag in den Warenkorb legt, gefragt wird, ob er ebenfalls das zweite Produkt erwerben möchte. Das Ergebnis der Analyse sind demnach temporale Komplementärgüter, die es auch ermöglichen, Produktbundles darzustellen. Amazon[Ama03] wendet eine ähnliche Personalisierung bereits mit nicht-temporalen Produktbundles an.
Im Rahmen des Projekts kann dieser Ansatz so erweitert werden, dass zu jedem Produkt ein Komplementärgut im Bundle angeboten wird. Welche Güter komplementär zu einander sind, variiert von Tag zu Tag und wird durch diese Muster ausgedrückt.
Das Amazon-Beispiel ist nicht ideal, da eine temporale Erweiterung dort nicht vorhanden ist, doch ist es eines der wenigen im Internet auffindbaren Beispiele, das ansatzweise eine praktische Umsetzung der Kernidee darstellt.

17.1.4 Beispiel

Das kalendarische Muster „10.11.$*$: Kürbiskuchen, Truthahn“ bedeutet, dass am 10. November eines jeden Jahres Kürbiskuchen und Truthahn zusammen verkauft werden. Dieses, aus der Quelle [LNWJ01] stammende Beispiel spiegelt die Einkaufsgewohnheiten der US-Amerikaner zu Thanksgiving wieder und ist somit für die Personalisierung sehr wertvoll.

17.2 Temporales Clustering von Warenkorbdaten

Mit diesem Ansatz soll probiert werden, die Warenkorbdaten temporal zu clustern. Ziel ist es, die Vorteile des Clusterings mit den Vorteilen der temporalen Analyse zu kombinieren. Es kann für dieses

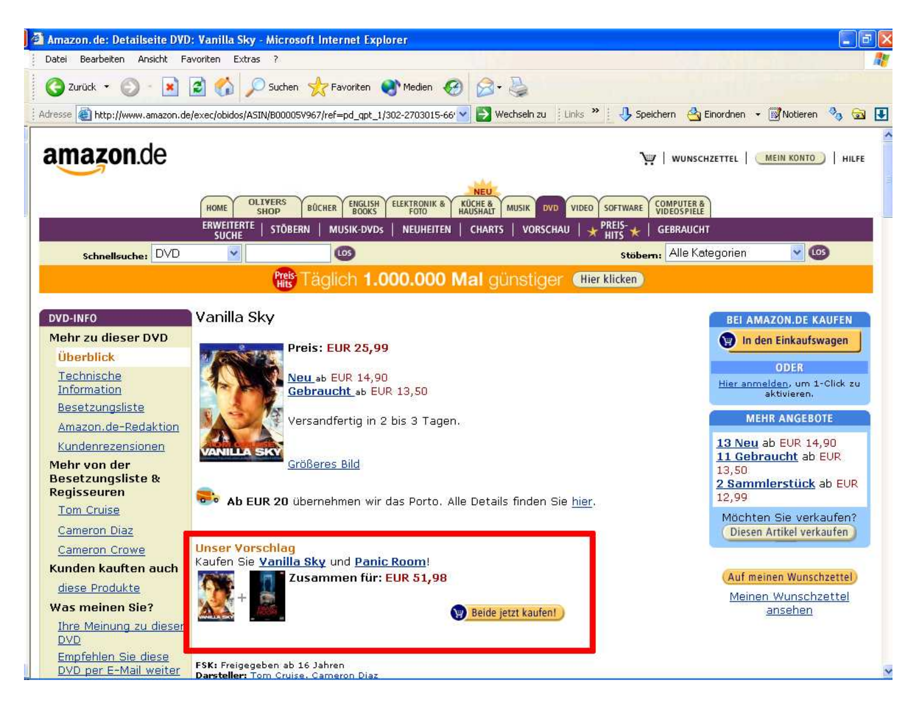


Abbildung 17.1: Nicht-temporale Produktbundles bei Internethändler Amazon [Ama03]

Verfahren ein beliebiger Clusterer verwendet werden.

Zur Vorgehensweise: Es werden alle Artikel und der Zeitpunkt des Einkaufs geclustert. Als Ergebnis werden Cluster erwartet, die Produkte gruppieren, die im selben Zeitraum gekauft werden. Die Größe des Zeitraums wird dabei durch den Clusterer bestimmt. Das Ziel dieser Analyse ist die Entdeckung saisonaler Produkte, also Produkte, die vorwiegend in einem bestimmten Zeitraum verkauft werden.
Die gewonnenen Cluster können relativ einfach für die Personalisierung im Online-Shop verwendet werden. Werden die Cluster beispielsweise so erstellt, dass sie einen Bereich von mehreren Stunden umfassen, z.B. Cluster 1 = 13 - 16 Uhr, können im Online-Shop während dieser Zeit Produkte vorgeschlagen werden, die zu dieser Zeit vorwiegend eingekauft werden. Die Qualität der Personalisierung lässt sich ggf. erhöhen, indem das Clustering auf Produkte mit einer gewissen

Häufigkeit beschränkt wird.

17.2.1 Ausgangsdaten

In den Datenbanken des Kartenanbieters und des Online-Händlers liegen Warenkorbdaten vor. Jeder Warenkorb besteht aus einer beliebigen Anzahl von Artikeln sowie den temporalen Attributen, die das Transaktionsdatum des gesamten Warenkorbs bestimmen. Mit Hilfe des Gültigkeitszeitsanfang (GZA) sollte bekannt sein, wann welcher Warenkorb zusammengestellt bzw. gekauft wurde. Daraus lässt sich ableiten, wann die enthaltenen Artikel gekauft wurden. Im Sinne des bitemporalen Datenmodells müsste für die folgenden Analysen die GZA verwendet werden. Da die Testdaten jedoch keine brauchbaren Werte vorhalten, muss auf die Transaktionszeit ausgewichen werden. Für die Analyse bedeutet dies keinen Nachteil: Die Transaktionszeit bestimmt, wann der Datensatz in der Datenbank gültig ist[Wie03]. Transaktionsanfang ist demnach die Zeit, zu dem der Warenkorb im Online-Shop angelegt wurde. Für die Analysen wird davon ausgegangen, dass der Warenkorb mit dem Startwert der Transaktion auch in der realen Welt gültig ist. Im Testszenario bzw. dem rudimentären Online-Shop sollte die Gültigkeitszeit korrekt gesetzt werden, so dass diese von späteren Analysen verwendet werden kann.
Um die nötigen Daten für die Analyse zu erhalten, ist ein Join aus zwei Entitäten notwendig. Beim Online-Händler sind dies die Entitäten *Warenkorb* und *Produkt_Warenkorb*.

17.2.2 Analyseverfahren

Als Analyseverfahren wird das Clustering verwendet, der ausgewählte Algorithmus heißt „XMeans“.

17.2.3 Interpretation

Als Ergebnis werden mehrere Cluster erwartet, denen eine beliebig große Menge von Produkten zugeordnet sind. Ein Cluster ist durch einen Centroid gekennzeichnet, um den herum die einzelnen Produkte gruppiert sind. Da es sich um ein temporales Clustering handelt, die Produkte also aufgrund ihrer temporalen Attribute gruppiert werden, steht jeder Cluster für einen Zeitraum. Der Zeitraum wird durch das

Kaufdatum des zuerst gekauften und des zuletzt gekauften Produktes eingegrenzt. Ein solcher Zeitraum kann als „Saison“ interpretiert werden. Produkte, die dann in diesem Cluster enthalten sind, haben als Eigenschaft, dass sie bevorzugt in dieser Saison gekauft werden. Diese Produkte werden deshalb als „saisonale Produkte“ bezeichnet.

17.2.4 Beispiel

Mögliche Ergebnisse sind Cluster, die

- nur Weihnachtsartikel
- nur Sommerartikel
- nur Winterartikel

enthalten.

17.3 Sequentielle Muster

Sequentielle Muster sind mit Assoziationsregeln zu vergleichen. Assoziationsregeln stellen Relationen und Zusammenhänge zwischen einzelnen Objekten oder deren Attributen her. Sequentielle Muster erfüllen eine ähnliche Aufgabe, doch in einem Punkt unterscheiden sie sich von den Assoziationsregeln gänzlich. Assoziationsregeln sind so genannte *Intra-Transaktionsmuster*, also Muster innerhalb einer Transaktion, beispielsweise innerhalb eines Warenkorbs. Sequentielle Muster dagegen sind *Inter-Transaktionsmuster*, das heißt, es werden Muster zwischen verschiedenen Transaktionen und nicht zwangsläufig innerhalb dieser untersucht. So können beispielsweise Zusammenhänge bei verschiedenen Einkäufen über einen Zeitraum von mehreren Wochen entdeckt werden (siehe auch [Wie03]).

17.3.1 Ausgangsdaten

Die Analyse der sequentiellen Muster wird auf den Warenkorbdaten des Online-Händlers durchgeführt.

17.3.2 Analyseverfahren

Als Analyseverfahren könnte der „Generalized Sequential Pattern“-Algorithmus (GSP) ausgewählt werden. Dieser Algorithmus ist als

temporale Erweiterung des in WEKA implementierten „Apriori“ zu verstehen [Wie03].

17.3.3 Interpretation

Sequentielle Muster stellen Zusammenhänge zwischen mehreren Transaktionen dar. Ein Muster kann so interpretiert werden, dass die enthaltenen Produkte in Folge gekauft werden und zwischen den Käufen ein bestimmter Zeitraum liegt. Im Online-Shop können aufgrund dieser Zusammenhänge Empfehlungen an den Besucher abgegeben werden. Wer das erste Produkt des Musters erworben hat, kann vom Online-Shop auf das Folgeprodukt hingewiesen werden. Dabei kann der Zeitraum, der zwischen den beiden Produkten liegt, vollständig abgewartet werden oder die Bewerbung des Folgeprodukts vorher stattfinden. Hierdurch kann der Zeitraum zukünftig vielleicht sogar verkürzt werden bzw. Folgekäufe beschleunigt werden!

17.3.4 Beispiele

Ein sequentielles Muster kann z.B. „$(B) \mapsto (C)$ mit einer Unterstützung von 30%“ lauten. In der Praxis ergeben sich dann Zusammenhänge wie z.B. „30% aller Käufer von Dieter Bohlens Buch „Nichts als die Wahrheit“ (B) kaufen innerhalb eines Monats auch die CD „Greatest Hits“ (C) von Dieter Bohlen“.

18 Anwendung temporaler Analysen

Dieses Kapitel zeigt die Anwendung der im vorangegangenen Abschnitt dargestellten temporalen Analysen auf (siehe Kapitel 17). Es werden Konfiguration und Ergebnisse der kalendarischen Muster sowie des temporalen Clusterings gezeigt.

18.1 Kalendarische Muster

Nach Abschluss der Implementierung des Algorithmus zum Finden kalendarischer Muster wurden verschiedene Analysen gestartet. Im Folgenden wird die Konfigurationsdatei sowie das damit erzielte Ergebnis beispielhaft erläutert.

18.1.1 Konfiguration und Parameter

Es wurde u.a. die folgende Konfigurationsdatei verwendet.

```
source=\
diko.framework.input.ConfigurableJDBCSourceIterator
relation=blubber
source.jdbcDriver=oracle.jdbc.driver.OracleDriver
source.jdbcConnection=\
jdbc:oracle:thin:@power2.offis.uni-oldenburg.de:\
1521:power2

source.jdbcLogin=haendler1
source.jdbcPassword=xxxxxxxx
source.jdbcSQL= SELECT DISTINCT w.wk_id, w.tza, w.tze,
            w.gza, w.gze, p.p_id \
            FROM warenkorb w, produkt_warenkorb p \
            WHERE w.wk_id = p.wk_id\
            AND rownum<=1000000 \
```

```
            AND w.wk_id>4 order by w.tza asc

source.tza=1
source.tze=2
source.gza=3
source.gze=4

attributeName_0= k_id
attributeType_0= numeric
attributeName_1= p_id
attributeType_1= nominal

sink.jdbcDriver= oracle.jdbc.driver.OracleDriver
sink.jdbcConnection=
jdbc:oracle:thin:@power2.offis.uni-oldenburg.de:\
        1521:power2
sink.jdbcLogin=haendler1
sink.jdbcPassword=xxxxxxxx
sink.jdbcTable=michi2
sink.jdbcIgnoreAttributeNames=1

algorithm=\ diko.framework.algorithms.\
TemporalAssociationRuleController
tempAssociations.minDate=01.01.2004
tempAssociations.maxDate=02.01.2000
tempAssociations.minSupport=0.2
tempAssociations.fuzzySupport=0.2
```

Die Parameter der obigen Konfiguration sind ausführlich in der Dokumentation des Frameworks (siehe Kapitel 14, Seite 107) erläutert. Deshalb soll im Folgenden lediglich auf die für diese Analyse relevanten Werte eingegangen werden.
Der Parameter „source.jdbcSQL=“ enthält den SQL-Befehl, der einen Join über die Entitäten *Warenkorb* und *Produkt_Warenkorb* bildet. Dabei werden die Attribute *wk_id* (Warenkorb-ID) und *p_id* sowie die temporalen Attribute des Warenkorb selektiert. Weitere wesentliche Optionen sind *DISTINCT*, um doppelte Datensätze auszuschließen und *rownum*<= 1000000, um die Gesamtanzahl auf eine Million zu begrenzen.

Mit „source.tza=“ und den folgenden drei Parametern werden die Spalten des SQL-Statements genannt, die die temporalen Attribute enthalten. Die Spalten sind von „0“ an durchnummeriert; „1“ steht also für die zweite Spalte.
Mit *attributeName* und *attributeType* werden Name und Typ der Attribute bestimmt. Im obigen Beispiel wurden die gleichen Namen wie in der Datenbank verwendet.
Weitere für diesen Algorithmus relevante Parameter sind *tempAssociations.minDate* und *tempAssociations.maxDate*, die den Zeitraum bestimmen, in dem die für die Analyse relevanten Transaktionen liegen müssen sowie *tempAssociations.minSupport*. Letzterer Parameter bestimmt den minimalen Support der Muster, die gefunden werden sollen. Ist das *minDate* größer als das *maxDate* wird vom Algorithmus automatisch von der ersten Transaktion bis zur letzten Transaktion analysiert.

18.1.2 Ergebnisse

Der folgende Ausschnitt zeigt die Ergebnisse, die mit der obigen Konfiguration erzeugt wurden. Das Format ist „Jahr, Monat, Tag, Support, Produkt“. Die Werte Jahr, Monat und Tag stehen für das Datum, an dem das gefundene Muster Gültigkeit hat. Für alle drei ist zudem ein Wert „-1“ möglich. Es handelt sich dann um ein „Starpattern“, wobei „-1“ als „Star“ (Stern bzw. *) zu verstehen ist und die oben (siehe Abschnitt 17.1.2) erläuterte Wildcard-Funktion ausübt: Für einen Stern kann ein beliebiger Wert verwendet werden. Im obigen Beispiel kann für das Jahr jedes Jahr im Analysezeitraum eingesetzt werden.
Mit „Support“ wird der Support des jeweiligen Musters angegeben und „Produkt“ enthält alle Artikel, die an diesem Datum gekauft werden.

Ergebnisvariation In mehreren Testläufen, die alle mehrere Stunden bis Tage dauerten, konnte die folgende Annahme bestätigt werden:
Je länger der Analysezeitraum, desto weniger Starpattern wurden gefunden. Die Erklärung liegt auf der Hand: Bei einem Zeitraum von einem Jahr muss ein Muster „25. Mai“ genau einmal vorkommen, um es in einem Starpattern „*,05,25“ darzustellen zu können. Je mehr Jahre der Zeitraum umfasst, desto häufiger muss das Pattern gefunden werden, um es als Starpattern darzustellen.

Jahr	Monat	Tag	Support	ID	Produkt
-1	-1	1	.25	1	2038
-1	-1	3	.25	2	241
-1	-1	4	.25	3	4882
-1	-1	4	.25	4	4883
-1	-1	4	.25	5	4884
-1	-1	4	.25	6	3580
-1	-1	4	.25	7	4901
-1	-1	4	.25	8	6901
-1	-1	4	.25	9	743
-1	-1	4	.25	10	6902
-1	-1	4	.25	11	679
-1	-1	6	.25	12	3720
-1	-1	6	.25	13	1751
-1	-1	6	.25	14	986
...	...	...	...	...	...

Tabelle 18.1: Kalendarische Muster

Darüber hinaus wird das Analyseergebnis durch den Mindest-Support beeinflusst. Je höher der gewünschte Support, desto weniger Muster werden gefunden.

18.2 Temporales Clustering von Warenkorbdaten

Das Ziel, stundengenaue Cluster zu finden, kann nicht umgesetzt werden, da die temporalen Attribute nur taggenau gespeichert sind. Es erfolgt deshalb während der Umsetzung die Einschränkung auf Cluster, deren kürzester Zeitraum genau einen Tag beträgt. Bei der Selektion der Daten aus der Datenbank werden die temporalen Werte so transformiert, dass sie die Anzahl der Tage seit Jahresbeginn angeben. Für das Clustering können so vor allem saisonale Zusammenhänge aufgedeckt werden.

18.2.1 Ergebnisse

Logs des temporalen Clusterings: Die Parameter sind ausführlich in Abschnitt 16.3.2 dokumentiert. Die Ergebnisse werden durch Veränderung der Parameter nicht wesentlich beeinflusst und können wie folgt erläutert werden.

```
XMeans.distanceFunction = weka.core.NominalDistance
XMeans.maxIterations = 5
XMeans.maxKMeans = 1000
XMeans.maxKMeansStructure = 1000
XMeans.minNumClusters = 5
XMeans.maxNumClusters = 100
XMeans.distanceValue = 1.0
XMeans.cutOffFactor = 0.5

Cluster centers                 : 2 centers

Cluster 0
            266.62068965517244
Cluster 1
            90.91404612159329

Distortion: 122.654358
```

Erläuterung Der Clusterer ermittelt immer die minimale Anzahl von möglichen Clustern. Die einzelnen Transaktionen werden gleichmäßig auf alle Cluster verteilt. Dies entspricht nicht den Erwartungen, kann aber damit begründet werden, dass die Testdaten zufällig erzeugt und dabei gleichmäßig auf das ganze Jahr verteilt wurden.
Aufgrund dieser Verteilung wurden die Testdaten nachträglich verändert. Nach Auswahl einiger Produkte wurden die Transaktionszeiten aller Warenkörbe, in denen diese Produkte enthalten sind, so angepasst, dass bestimmte Produkte in bestimmten Monaten gekauft wurden. Mit diesen Voraussetzungen erzeugt die Analyse das gewünschte Ergebnis: Es werden saisonale Produkte gefunden. In der Praxis könnten dies beispielsweise Weihnachtsartikel sein, die ausschließlich oder vorzugsweise in der Vorweihnachtszeit erworben werden. Weitere Beispiele für saisonale Produkte sind Bademoden, Winterbekleidung, Saisonfrüchte, etc.

18.2.2 Bewertung

Das temporale Clustering eignet sich sehr gut, um saisonale Zusammenhänge festzustellen. Produkte, die nur zu bestimmten Zeiten ge-

kauft werden, bilden einen eigenen Cluster und lassen sich im Folgejahr verkaufsfördernd bewerben.
Vorsicht gilt bei saisonalen Produkten, deren Cluster nur wenige Tage umfasst. Hierbei könnte es sich um einen Feiertag handeln, der im Folgejahr möglicherweise auf ein anderes Datum fällt. Die Bewerbung der saisonalen Produkte dieses Clusters könnte so unter Umständen zu früh oder - noch schlimmer - zu spät erfolgen.
Ein weiteres Risiko besteht bei Angebotswaren. Wenn der Händler beispielsweise einen Sonderposten zu einem bestimmten Zeitraum anbietet, könnte sich dieser nach der Analyse eventuell in einem eigenen Cluster wiederfinden. Das Bewerben eines bereits ausverkauften Sonderpostens vom Vorjahr wäre sinnlos für das Unternehmen.

18.2.3 Erweiterungsmöglichkeiten

Es gibt eine Vielzahl von denkbaren Erweiterungen. Eine wesentliche Erweiterung, die das Analyseergebnis mit wenig Aufwand qualitativ verbessern würde, wäre die Vorverarbeitung der Transaktionen, z.B. durch die Reduzierung der Analyse auf wenige Produkte oder auf häufig gekaufte Produkte.

19 Ergebnisse und Schlussbetrachtung der Personalisierung

Dieses Kapitel stellt mit einem Soll-Ist-Vergleich die Aufgaben der Personalisierung sowie deren Erfüllung dar, zählt dann die entsprechenden Meilensteine auf und schließt mit einem Fazit.

19.1 Soll-Ist-Vergleich

Im Rahmen des Soll-Ist-Vergleiches wird dargestellt, welche in den Kapiteln 15 und 17 aufgeführten Analysen umgesetzt worden sind. Des Weiteren enthält der Soll-Ist-Vergleich eine Übersicht, welche der Analysen anwendbare Ergebnisse für die Personalisierung des Online-Shops liefern.

19.2 Meilensteine



Nr	Meilenstein	Start	Ende	Dauer	Q4 02			Q1 03			Q2 03			Q3 03		
					Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep
1	Personalisierungskonzept	09.06.2003	20.07.2003	6w												
2	PG-freie Zeit	21.07.2003	27.07.2003	1w												
3	Personalisierungskonzept II	28.07.2003	31.08.2003	5w												

Abbildung 19.1: Meilensteine Personalisierung

Der Meilenstein „Personalisierungskonzept“ dauert von 09.06.03 bis 31.08.03. In der Abbildung 19.1 erfolgt die Darstellung in drei getrennten Abschnitten, da der eigentliche Meilenstein durch die projektfreie Zeit (21.07.-27.07.03) unterbrochen wird. Insgesamt stehen der Aufgabe Personalisierungskonzept 11 Wochen zur Verfügung. Ursprünglich wurden nur sechs Wochen eingeplant, da die weitreichen-

Analyse	Umsetzung	Anwendung
Clusteranalyse nach Kunden	ja	zwei Top5-Tabellen
Clusteranalyse nach Kunden und gekauften Produkten	ja	nicht möglich
Assoziationsanalyse (nicht-temporal)	nein	-
kombinierte Analysen	nein	
weitere Möglichkeiten des Clusteralgorithmusses	nein	-
kalendarische Muster (temporal)	ja	temporale Produktbundles
temporales Clustering	ja	saisonale Produkte
sequentielle Muster	nein	-

Tabelle 19.1: Soll-Ist-Vergleich

den Konsequenzen und vor allem mit dieser Aufgabe in Verbindung stehenden Einzeltätigkeiten (u.a. Durchführung von Analysen) nicht vollständig berücksichtigt wurden, war eine Anpassung nötig.
Abschließend läßt sich sagen, dass auch die Zeit von 11 Wochen sehr knapp bemessen war, da die Analyse einerseits durch die Vielzahl der möglichen Parameter und andererseits durch die lange Laufzeit der einzelnen Durchgänge viel Zeit beanspruchte.

Meilenstein Personalisierung: *09.06.03* bis *20.07.03* sowie *28.07.03* bis *31.08.03*, Dauer: 11,0 Wochen.

19.3 Fazit

Im Rahmen der nicht-temporalen Analyse hat sich herausgestellt, dass die separate Verwendung eines Clusteralgorithmus wenig Aussagekraft in Bezug auf die Personalisierung hat. Es bietet sich daher eine Verbindung mit Assoziationsanalysen an, um aussagekräftigere Analyseergebnisse zu erhalten. Da jedoch die Personalisierung in Bezug auf temporale Analysen im Mittelpunkt steht, beschränkt sich das Personalisierungskonzept auf eine Auswahl der erarbeiteten nicht-

temporalen Analyseverfahren. Weitere Möglichkeiten des Clustering sind im Punkt 15.3 aufgeführt.
Die Untersuchungen der Personalisierungsgruppe haben ergeben, dass der aktuelle Stand des Algorithmus für kalendarische Muster wenig sinnvolle Anwendungsmöglichkeiten im Bezug auf die Personalisierung bietet. Um die kalendarischen Muster ausgiebig anwenden zu können sind daher die Erweiterungsmöglichkeiten Fuzzy-Match-Ratio und flexible Kalenderschemata vorgestellt worden. Zusätzlich soll im weiteren Verlauf der Projektgruppe noch ein zweiter Algorithmus (Sequentielle Muster) implementiert werden, um den eingeschränkten Anwendungsbereich der kalendarischen Muster zu erweitern.
Neben den verwendeten Algorithmen sind zudem noch weitere Ausrichtungsmöglichkeiten des Angebotes auf den Kunden und die persönlichen Ansprache auf der Internetseite des Online-Shops näher untersucht worden. Es ist jedoch entschieden worden, dieses nicht weiter auszuführen, da es nicht direkt zum Personalisierungskonzept im Rahmen der Projektgruppe gehört.

Teil V

Ergebnisse und Schlussbetrachtung

20 Shop

Um die von der Projektgruppe erarbeitete Personalisierungsbibliothek testen zu können und deren Ergebnisse zu visualisieren, ist ein Online-Shop-System aufgesetzt worden. Dieses soll einem sich anmeldenden Nutzer anhand der Analyseergebnisse der Personalisierungsbibliothek und der aus den Händler-Szenarios stammenden demographischen Daten ein speziell auf den Nutzer zugeschnittenes, personalisiertes Warenangebot bieten.

20.1 Anforderungen

Die Hauptaufgabe des Online-Shops ist es, die von der Personalisierungsbibliothek gefundenen Personalisierungen zu visualisieren. Weiterhin sollen jedoch Grundfunktionalitäten eines Online-Shops wie z.B. ein Benutzer-Management, Produktübersicht- und Produktdetailseiten sowie ein Warenkorb implementiert werden, um den Online-Shop realer wirken zu lassen.

20.2 Implementierung

Es wurde entschieden, den Online-Shop mittels jsp-Technik zu implementieren. Hierzu wurde ein *Tomcat-Server* der *Apache-Jakarta Group* auf dem Rechner *power2.offis.uni-oldenburg.de* aufgesetzt, der seine Dienste auf Port *8077* anbietet. Daher ist der Online-Shop unter der URL *http://power2.offis.uni-oldenburg.de:8077/* zu erreichen. Die Tomcat-Serversoftware steht unter der Apache-License und ist somit frei nutzbare Open-Source Software.
Der Online-Shop basiert auf dem Datenbankschema des Händler 1. Dies hat den Vorteil, von Anfang an auf den Betrieb eines Online-Shops ausgelegt zu sein. Es bietet daher alle dafür notwendigen Entitäten, wie z.B. Warenkörbe, Produkt-Kategorien usw. Außerdem wurde die Datenbank von Händler 1 mit vielen Testdaten gefüllt, so dass hiermit problemlos ein Online-Shop betrieben werden kann.

Bei der Gestaltung der Seiten wurde entschieden, sämtliche Online-Shop-Funktionalitäten in die Basisseite *index.jsp* einzubinden, die per URL-Parameter konfigurierbar ist. In Abhängigkeit des Wertes des Parameters *CMD* wird eine der fünf folgenden Seiten geladen:

- *content/root.jsp*: Startseite mit DIKO-Logo und den Top-Level-Kategorien
- *content/catalog.jsp*: Darstellung eines Kataloges mit seinen untergeordneten Kategorien und Produkten
- *content/product.jsp*: Darstellung der Produktdetailseite mit weiterführenden Informationen zu einem speziellen Produkt
- *content/basket.jsp*: Warenkorb zur Aufnahme von Produkten aus dem Online-Shop
- *content/user.jsp*: Darstellung aller bekannten demographischen Daten eines angemeldeten Benutzers

Weiterhin können den einzelnen Unterseiten Parameter per URL übergeben werden, die das Verhalten der Unterseiten bestimmen, was in Abschnitt 20.2.2 erläutert wird.

20.2.1 Benutzer-Management

Für das Benutzer-Management wurde eine Java-Klasse implementiert, die die Attribute eines Kunden aus der Datenbank abbildet. Ein Benutzer kann sich mit seinem Nutzernamen, welcher der KundenID aus der Datenbank entspricht, und seinem Kennwort, welches ebenfalls in der Datenbank gespeichert ist, über ein Anmeldeformular auf allen Seiten des Online-Shops anmelden. Stimmen die Angaben von Nutzernamen und Kennwort mit einem Datensatz eines Kunden aus der Datenbank überein, so werden alle demographischen Daten des Nutzers in das User-Objekt geladen. Anschließend wird das User-Objekt in die Session gelegt, so dass der Benutzer auch auf folgenden Seiten eindeutig zugeordnet werden kann; selbst im Falle des Verlassens des Online-Shops und einem anschließenden erneuten Besuch.

20.2.2 Darstellung der Seiten

Wie bereits oben erwähnt, erfolgt die Navigation durch den Online-Shop über eine zentrale jsp-Seite: *index.jsp*.

Diese Seite enthält eine Tabelle, in die verschiedene Unterseiten, die als eigenständige jsp-Dateien auf dem Server abgelegt sind, eingebunden werden. Die Aufteilung der Seite ist dem Schema in Abbildung 20.1 zu entnehmen. Zudem werden noch zwei weitere jsp-Dateien ein-

<table>
<tr><td>header.jsp</td><td rowspan="2">{root, ctalog, product, user, basket}.jsp

* in Abhängigkeit des übergebenen Paramters CMD wird eine der Seiten dargestellt.

* alle diese Dateien sind im Verzeichnis /content/</td></tr>
<tr><td>navigation.jsp</td></tr>
<tr><td>footer.jsp</td><td>footer2.jsp</td></tr>
</table>

Abbildung 20.1: Aufbau des Online-Shops

gebunden: *logic.jsp* und *final.jsp*. Diese Seiten kapseln Funktionalität und stellen keine Inhalte dar. Im Folgenden werden die einzelnen jsp-Seiten näher beschrieben.

logic.jsp

Diese Datei wird als erste Datei in die *index.jsp* eingebunden, noch bevor etwas dargestellt wird. In dieser Datei wird zunächst eine Datenbankverbindung zur Datenbank des Händler 1 aufgebaut. Anschließend wird ein User-Objekt instantiiert und, falls die Parameter *user* und *password* mit Werten belegt sind, d.h. sich ein Nutzer anmeldet, mit den Daten des sich anmeldenden Nutzers gefüllt. Weiterhin wird der Warenkorb aus der Session geholt, falls einer existiert. Andernfalls wird ein neuer Warenkorb instantiiert. Abschließend wird der Parameter *CMD* ausgewertet. Dies geschieht anhand einer Integer-Konstanten, anhand der in der *index.jsp* entschieden wird, welcher Inhalt eingebunden wird.



Abbildung 20.2: Darstellung von *header.jsp*

header.jsp

Die Datei *header.jsp* wird als zweite Datei von der Datei *index.jsp* eingebunden. Sie beinhaltet das kleine DIKO-Logo und die Überschrift *Shop*, jedoch keine dynamischen Inhalte.

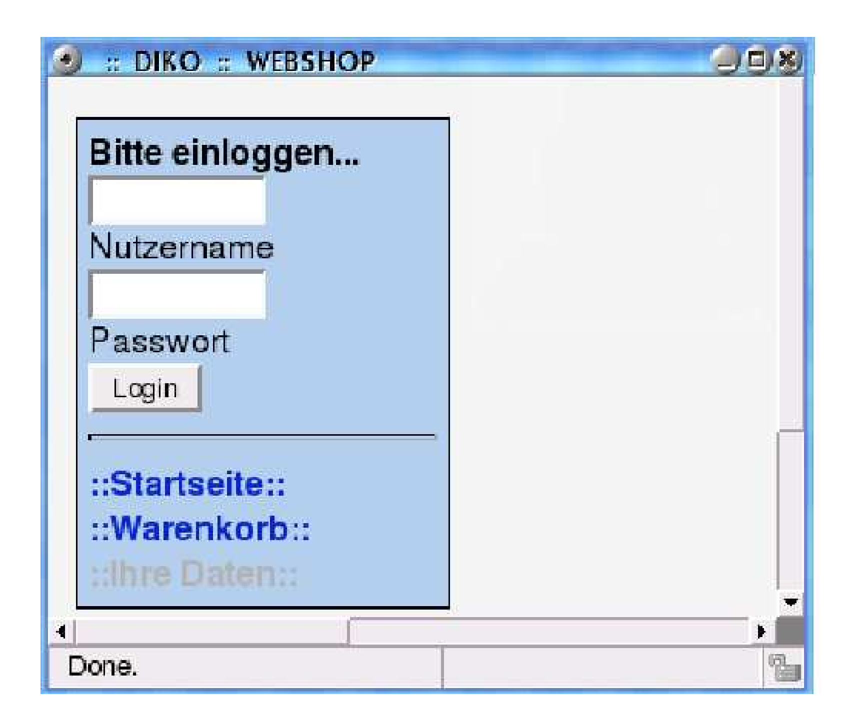


Abbildung 20.3: Darstellung von *navigation.jsp* ohne angemeldeten Benutzer

navigation.jsp

Diese Datei ermöglicht die Navigation durch die Seiten des Online-Shops. Es existiert eine Box mit einem Login-Formular und einem Login-Button, falls kein Nutzer angemeldet ist. Weiterhin befinden sich hier Links auf die Startseite und den Warenkorb.



Abbildung 20.4: Darstellung von *navigation.jsp* mit angemeldetem Benutzer

Meldet sich ein Benutzer an, so erscheint eine persönliche Begrüßung mit Nennung des Namens des angemeldeten Nutzers und ein Logout-Button. Weiterhin erscheint unter den beiden bereits vorhandenen Links ein weiterer mit dem Titel *Ihre Daten*, der zur Darstellung der Seite *user.jsp* führt.

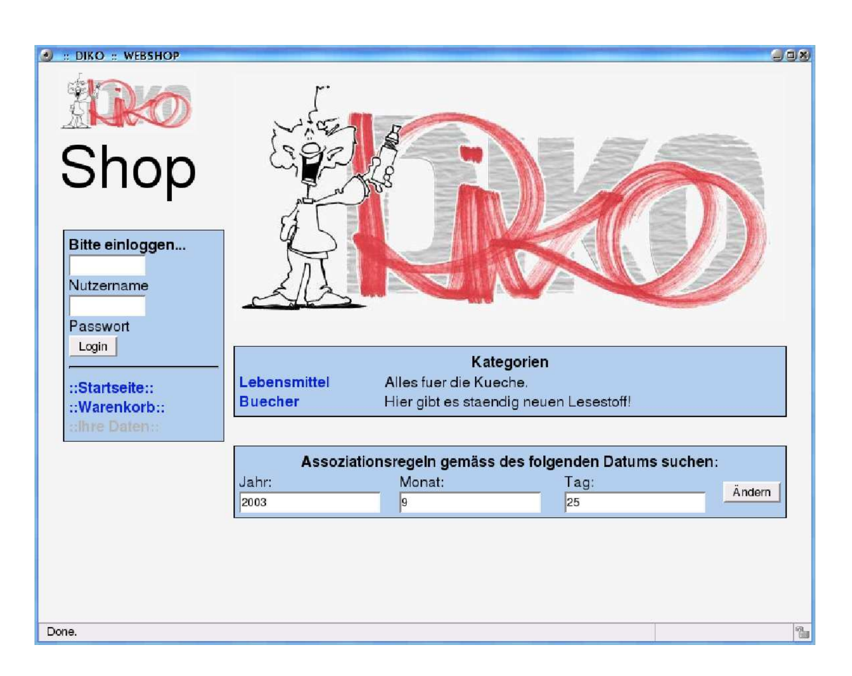


Abbildung 20.5: Darstellung von *root.jsp*

root.jsp

Diese Datei stellt das DIKO-Logo in großem Format dar, um den Online-Shop mit dem DIKO-Projekt in Verbindung bringen zu können. Darunter wird eine Tabelle dargestellt, die Links zu den verschiedenen Produkt-Kategorien beinhaltet.

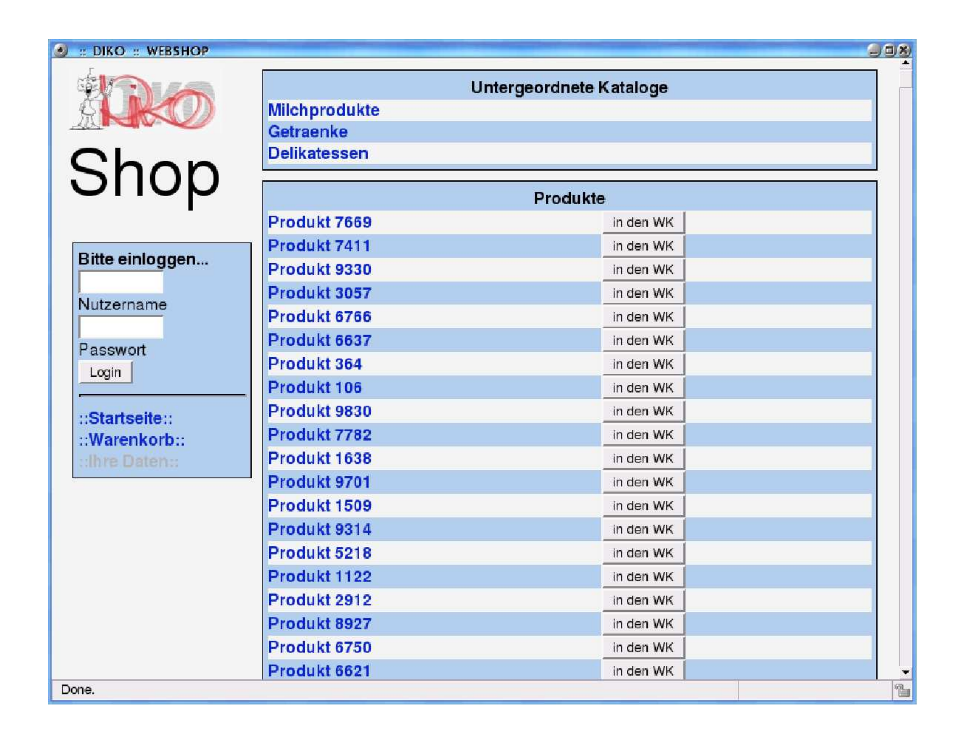


Abbildung 20.6: Darstellung von *catalog.jsp*

catalog.jsp

Diese Datei stellt einen Produktkatalog dar. Dieser wird über die Parameter *kId* und *kSid* spezifiziert. Ein Katalog kann untergeordnete Kataloge beinhalten, die in der oberen Tabelle verlinkt dargestellt sind. Das Folgen eines Kataloglinks führt zum untergeordneten Katalog.
Ein Katalog kann auch Produkte beinhalten, die in der unteren Tabelle verlinkt dargestellt sind. Rechts neben einem Produktlink wird ein Link *In den WK* dargestellt. Durch Folgen dieses Links kann ein Produkt in den Warenkorb aufgenommen werden. Das Folgen eines Produktlinks führt zur Produktdetailseite *product.jsp*.

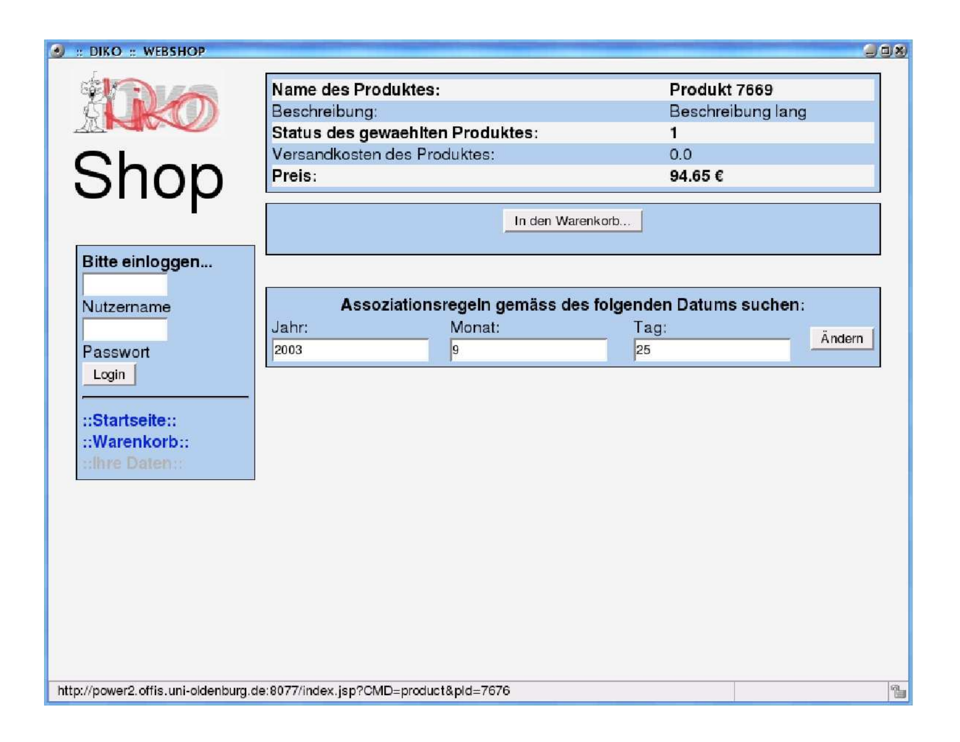


Abbildung 20.7: Darstellung von *product.jsp*

product.jsp

Diese Datei stellt die Details eines ausgewählten Produktes dar, welches per Parameter *pId* spezifiziert wird, die der ProduktID eines Produktes aus der Datenbank entspricht. Zu den Details gehören der Name, die Beschreibung, der Verfügbarkeitsstatus sowie die Versandkosten und der Verkaufspreis des Produktes. Unten auf der Seite wird ein Link dargestellt, der das Produkt in den Warenkorb einfügt.

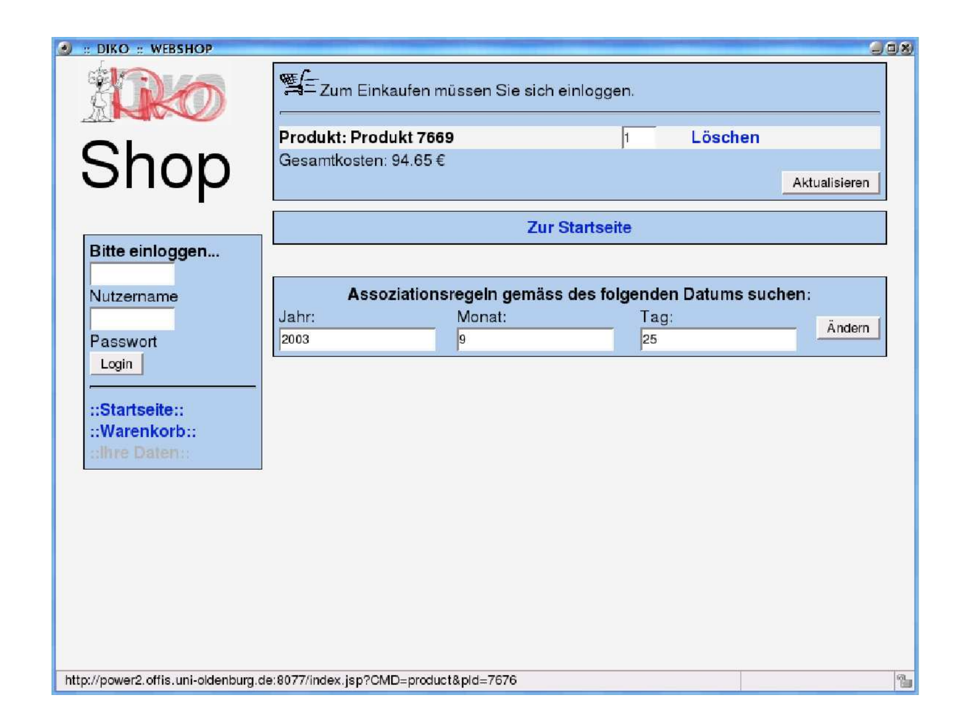


Abbildung 20.8: Darstellung von *basket.jsp*

basket.jsp

Die Datei *basket.jsp* ist für die Darstellung des Warenkorbs zuständig. Die Daten des Warenkorbs werden über die Klasse *diko.shop.Basket* verwaltet. Zu Beginn einer Session wird ein neues Objekt dieser Klasse angelegt und in der Session gespeichert. Auf der Warenkorb-Seite werden dann die im Basket enthaltenen Produkte dargestellt. Jedes Produkt wird mit seinem Namen, der momentan im Warenkorb vorhandenen Anzahl und einem Link zum Entfernen des Produktes aus dem Warenkorb dargestellt. Die Anzahl der im Warenkorb befindlichen Produkte wird in einem Formularfeld dargestellt, mit dem sie sich manipulieren lässt. Außerdem werden die Kosten des gesamten Warenkorbs dargestellt.



Abbildung 20.9: Darstellung von *user.jsp*

user.jsp

Diese Datei stellt die demographischen Daten eines Nutzers dar. Hierzu zählen Name, Vorname, Geburtsdatum, Anrede, Adresse, E-Mail, Telefon und die Gültigkeitszeitstempel. Die Daten werden zur Aufrufzeit der Seite aus dem User-Objekt aus der Session geladen.
Ist kein Nutzer bei Aufruf dieser Seite angemeldet, so erfolgt die Ausgabe eines Hinweises, dass ein Nutzer sich anmelden soll.

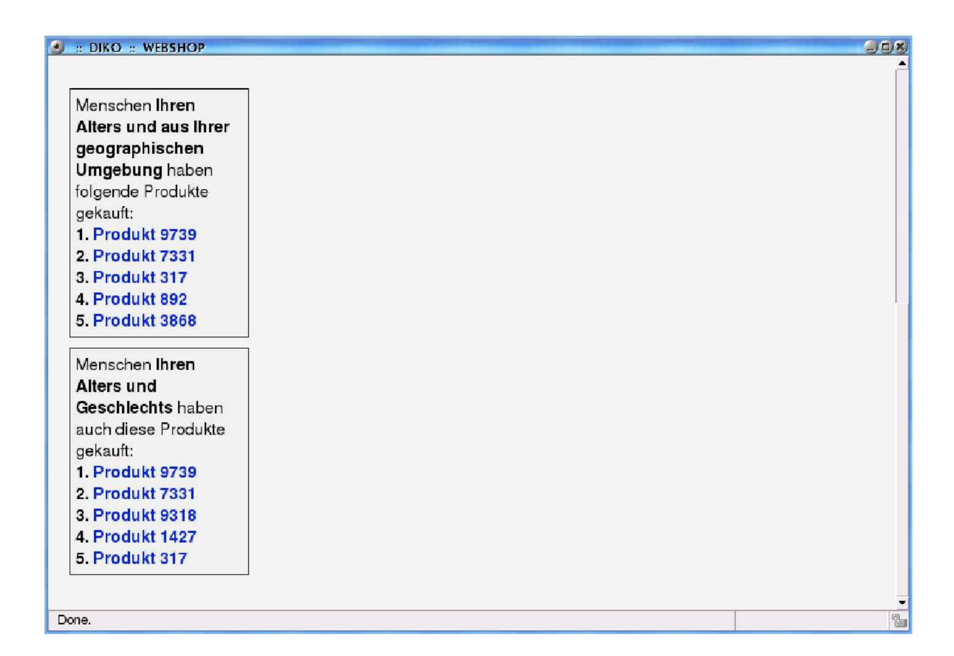


Abbildung 20.10: Darstellung von *footer.jsp*

footer.jsp

Diese Datei beinhaltet die personalisierten Angebote, die einem angemeldeten Nutzer spezifisch dargeboten werden. Ist kein Nutzer angemeldet, so wird durch diese Datei kein Inhalt ausgegeben.



Abbildung 20.11: Darstellung von *footer2.jsp*

footer2.jsp

Diese Datei beinhaltet die zum Tag des Seitenaufrufes passenden Angebote und ein Formular, um die aktuellen Tag manipulieren zu können. Siehe dazu auch Abschnitt 20.4.2.

final.jsp

Diese Datei stellt, wie schon die Datei *logic.jsp*, eine Kapselung von Funktionalität dar und beinhaltet keinen Code zur Darstellung von Objekten auf der Website. Sie wird als letztes in die Datei *index.jsp* eingebunden. Im Falle des Abmeldens eines Nutzers wird hier die Session gelöscht.

20.3 Anwendungsfall

Ein Benutzer, der auf die Seite geleitet wird, bekommt zunächst die Startseite zu sehen. Auf dieser werden verschiedene Elemente dargestellt: oben links wird die Datei *header.jsp* eingefügt, die das DIKO-Logo in kleiner Form sowie das Wort *Shop* darstellt. Darunter erscheint der Inhalt der Datei *navigation.jsp*, die das Formular zur Eingabe der Benutzerdaten in einem Kasten darstellt, sofern der Benutzer nicht schon angemeldet ist. Im Content-Bereich der Startseite erscheint der Inhalt der Datei */content/root.jsp*, der das DIKO-Logo in großer Form und eine Tabelle mit allen Kategorien anzeigt, die in der Datenbank vorhanden sind.
Meldet sich der Benutzer mit einer passenden Kombination aus Benutzername und Passwort am System an, so ändert sich lediglich

der Inhalt des Kastens. Anstelle des Formulars zur Eingabe von Benutzernamen und Passwort erscheint nun eine Willkommensmeldung mit Vornamen und Nachnamen des angemeldeten Benutzers sowie ein Logout-Button.
Der Benutzer hat die Möglichkeit, eine der Kategorien anzuwählen oder seine demographischen Daten einzusehen. Klickt der Benutzer beispielsweise auf die Kategorie *Lebensmittel*, so ändert sich der Inhalt des Content-Bereiches der Webseite. Es wird daraufhin anstelle der Datei *root.jsp* die Datei *catalog.jsp* eingebunden, die für die Darstellung von den der gewählten Kategorie untergeordneten Katalogen und den in der gewählten Kategorie enthaltenen Produkten zuständig ist. Hier kann der Benutzer dann entweder einen untergeordneten Katalog oder ein Produkt auswählen. Wählt der Benutzer beispielsweise das Produkt mit der Bezeichnung *Produkt 7669*, so wird die Produktdetailseite angezeigt, die von der Datei *product.jsp* erzeugt wird. Hier werden der Name des Produktes, eine nähere Beschreibung sowie der Verfügbarkeitsstatus , die Versandkosten und der Preis des Produktes dargestellt. Darunter erscheint ein Button mit der Aufschrift *In den Warenkorb...*, der das Produkt in den Warenkorb des angemeldeten Benutzers einfügt.
Nach Betätigung des Buttons erscheint der Warenkorb des Benutzers, in dem sich das soeben hinzugefügte Produkt bereits befindet. Der angemeldete Nutzer kann hier noch die Anzahl des zu kaufenden Produktes angeben. Nach Eingabe der Anzahl und Betätigung des Buttons mit der Aufschrift *Aktualisieren* werden die Gesamtkosten, die unter den Produkten aufgeführt sind, neu berechnet. Der angemeldete Benutzer hat anschließend die Möglichkeit, über den Link zur Startseite weiter im Katalog zu navigieren und weitere Produkte in den Warenkorb aufzunehmen.
Auf eine Implementation der restlichen Schritte zum Kauf der im Warenkorb befindlichen Daten wurde verzichtet, da diese für den Zweck des Online-Shops, der die Personalisierungsergebnisse der Personalisierungsbibliothek visualisieren soll, nicht notwendig sind.

20.4 Personalisierung

Der Online-Shop visualisiert die Ergebnisse der Personalisierungsbibliothek, die von der Projektgruppe implementiert wurde. So sollen alle Ergebnisse der Analysen, die mit Hilfe des Frameworks aus den

Analyse-Algorithmen aus WEKA und der temporalen Assoziationsanalyse auf den Testdaten der Händler- und Kartenanbieterdatenbanken erzeugt wurden, im Online-Shop in Form von dynamischen Webseiten visualisiert werden.
Die verschiedenen Formen der in den Online-Shop eingeflossenen Personalisierungstechniken werden im Folgenden vorgestellt.

20.4.1 Top5 der meistverkauften Produkte einer Nutzergruppe

Meldet sich ein Benutzer am Online-Shop an, so wird anhand seiner KundenID eine Einordnung bezüglich zweier Cluster durchgeführt. Die erste Einordnung basiert auf den gefundenen Clustern, wobei die Aspekte des Alters und der geographischen Herkunft anhand der Postleitzahl des Kunden berücksichtigt werden. Die zweite Einordnung basiert auf den gefundenen Clustern, die unter Berücksichtigung der Aspekte des Alters und des Geschlechts des angemeldeten Kunden gefunden wurden (siehe hierzu Abschnitt 16.1).
So werden mit der KundenID jeweils für beide Cluster eine ClusterID aus der Datenbank gelesen. Anhand dieser ClusterIDs können die fünf meistverkauften Produkte der anderen Kunden des jeweiligen Clusters aus der Datenbank geladen und in Form einer Liste unterhalb des Anmeldeformulars im Online-Shop dargestellt werden.
Direkt unterhalb des Anmeldeformulars werden die fünf meistverkauften Produkte der Kunden des Clusters, welches unter Berücksichtigung der Aspekte des Alters und der Postleitzahl erstellt wurden, und zu dem der angemeldete Kunde klassifiziert wurde, dargestellt.
Darunter erscheinen die fünf meistverkauften Produkte der Kunden des Clusters, welches unter Berücksichtigung der Aspekte des Alters und des Geschlechts gefunden wurde, und zu dem der angemeldete Kunde klassifiziert wurde, dargestellt.

20.4.2 Einbinden von Assoziationsregeln

Eine weitere wichtige Anwendung der Personalisierung im Online-Shop ist das Einblenden von Produkten, die dem Kunden zum Kauf empfohlen werden. Dies geschieht auf zwei grundsätzlich verschiedene Arten:

- Auf der Produktdetailseite werden dem Kunden Produkte angeboten, die gemäß den gefundenen Assoziationsregeln mit dem

dargestellten Produkt zusammen gekauft werden.

- Auf allen Seiten werden dem Kunden Produkte angeboten, die bevorzugt an dem aktuellen Tag gekauft werden.

Für beide Einblendungen wird die Klasse *diko.shop.AssociationRules* verwendet. Diese bietet zwei Methoden, um Assoziationsregeln aus der Datenbank zu laden.
Die Methode *loadRules(float minSupport)* erzeugt eine Liste von Produkten, die in Assoziationsregeln vorkommen, die mindestens den angegebenen Support erfüllen und zum aktuellen Datum gültig sind. Diese Methode wird benutzt, um die zur Einkaufszeit passenden Produkte auf allen Seiten des Online-Shops einzublenden.
Die Methode *loadRules(float minSupport, Product p)* arbeitet ähnlich, prüft jedoch zusätzlich, ob das angegebene Produkt in den beachteten Assoziationsregeln vorkommt. Diese Methode wird verwendet, um die zu einem auf der Detail-Seite dargestellten Produkt passenden Produkt-Empfehlungen darzustellen.
Da die verwendeten Assoziationsregeln nur an bestimmten Kalendermustern gültig sind, kann es aufgrund der Beschaffenheit der von der Projektgruppe erzeugten Testdaten vorkommen, dass an bestimmten Tagen keine Regeln gültig sind. Um dennoch an jedem beliebigen Tag das Testen des Online-Shops zu ermöglichen, verfügt dieser über ein Formular, mit dem das Datum, zu welchem die passenden Regeln gesucht werden, beeinflussbar ist. Standardmäßig wird hier das aktuelle Datum vorgegeben.

21 Soll-Ist-Vergleich vom Gesamtprojekt

Der folgende Abschnitt beinhaltet einen Soll-Ist-Vergleich des Gesamtprojektes auf Grundlage der Anforderungen des Gesamtszenarios.

21.1 Einleitung

In diesem Kapitel werden die Anforderungen des Gesamtszenarios hinsichtlich ihrer Zielerreichung überprüft. Es wird dabei detailliert erläutert, auf welche Weise die einzelnen Anforderungen erfüllt und warum einige Bereiche nicht oder nur zu Teilen bearbeitet worden sind. Als Grundlage für diesen Vergleich dient die Anforderungsdefinition für das Gesamtszenario (siehe Kapitel 2), an dessen Gliederung sich auch dieses Dokument orientiert. So werden zuerst die funktionalen Anforderungen (vgl. Abschnitt 2.2) unterteilt in die Bereiche Integration (Abschnitt 21.2), Analyse (Abschnitt 21.3) und Shop (Abschnitt 21.4) betrachtet, anschließend die nichtfunktionalen Anforderungen(siehe Abschnitt 21.5, vgl. Abschnitt 2.3) sowie die Benutzerschnittstellen und Fehlerverhalten (siehe Abschnitt 21.6, vgl. Abschnitt 2.4) und die Dokumentationsanforderungen (siehe Abschnitt 21.7, vgl. Abschnitt 2.5). Abschließend werden die Ergebnisse in dem Abschnitt 21.8 anhand der Abbildung 21.1 zusammenfassend beschrieben.
Den Abschluss bildet die Meilensteinplanung, die alle von der Projektgruppe vereinbarten Hauptaufgaben und ihre Fertigstellungstermine aufführt. Dabei wird jede Aufgabe kurz erläutert, ein Vergleich zum Meilensteinplan des Zwischenberichts hergestellt und im Fazit die Einhaltung bzw. Nicht-Einhaltung betrachtet.

21.2 Integration

Die Integration unterstützt, wie gefordert, den Datenimport der beiden Händlerdatenbanken in die Kartenanbieterdatenbank. Weiterhin sind jeweils Methoden zum initialen Datenimport für beide Händlerdatenbanken entwickelt worden. Das Preprocessing im Bereich der Integration umfaßt die Beseitigung ungültiger Einträge in den Transaktionsdaten (Offline-Händler) und die Auszeichnung aller Daten mit der jeweiligen Quelle (HaendlerID in allen Relationen der Kartenanbieterdatenbank).
Die Integration kann, wie im Szenario vorgesehen, losgelöst von der Analyse durchgeführt werden. Durch das Abspeichern eines Aktualisierungsdatums ist dafür gesorgt, dass nur die neuen bzw. veränderten Daten integriert werden. Ein bestimmter Aktualisierungsrhythmus ist nicht festgelegt worden, da bisher nur die angelegten Testdaten integriert, aber keine neuen Transaktionsdaten generiert worden sind. Die detaillierten Information sind in dem Integrationsabschnitt nachzulesen (siehe Kapitel II).

21.3 Analyse

In der Analyse sind, den Vorgaben entsprechend, jeweils ein nichttemporales und temporales Verfahren durchgeführt worden. Für den nichttemporalen Bereich ist das Clustering in verschiedenen Ausprägungen zur Anwendung gekommen: Im einzelnen sind es das Clustering nach Kundenattributen (siehe Abschnitt 16), das Clustering der Kunden nach gekauften Produkten (siehe Abschnitt 16.2) und das Clustering von Warenkörben nach Produkttypen (siehe Abschnitt 16.3). Die verschiedenen Verfahren greifen dabei alle auf die in WEKA vorhandenen Clusteralgorithmen zurück. Der temporale Bereich wird durch die kalendarischen Muster abgedeckt, für den ein eigener Algorithmus implementiert worden ist.
Ausprägungen zur Anwendung gekommen: Im Einzelnen sind es das Clustering nach Kundenattributen (siehe 16), das Clustering der Kunden nach gekauften Produkten (siehe 16.2) und das Clustering von Warenkörben nach Produkttypen (siehe 16.3). Die verschiedenen Verfahren greifen dabei alle auf die in WEKA vorhandenen Clusteralgorithmen zurück. Der temporale Bereich wird durch die kalendarischen Muster abgedeckt, für den ein eigener Algorithmus implementiert

worden ist.
Zur Steuerung der Analysen ist ein Framework entworfen und implementiert worden, wodurch ein metadatengesteuerter Ablauf ermöglicht wurde. Die notwendigen Angaben werden dem Framework über sogenannte Configfiles übergeben. Das Framework ermöglicht weiterhin, einer Analyse eine beliebige Anzahl von Vor- und Nachbearbeitungsschritten hinzuzufügen (Prä- und Postprozessoren) als auch mehrere Analysen nacheinander durchzuführen. Bei dem in der Anforderungsdefinition erwähnten Austauschformat handelt es sich um das in WEKA verwendete arff-Format.
Es ist darüberhinaus jedoch sowohl für den Import als auch für das Speichern der Analyseergebnisse möglich, direkt mit der Datenbank zu kommunizieren. Details zum Framework sind dem Kapitel 14 zu entnehmen.

21.4 Shop

Der Online-Shop ist zur Evaluierung der Analyseergebnisse erzeugt worden. Als Datengrundlage dient wie vorgesehen die Datenbank von Händler 1, dem Online-Händler. Im Shop sind die geforderte Darstellung der baumartigen Katalogstruktur, Detailseiten zu den einzelnen Produkten und eine Warenkorbfunktion realisiert worden. Die Vorlieben eines Kunden werden über zwei Top5-Listen für einen angemeldeten Kunden berücksichtigt, jedoch unabhängig von der aktuellen Position in der Katalogstruktur. Auf den Produktdetailseiten werden dem Kunden zutreffende temporale Assoziationsregeln zu dem gewählten Produkt in Abhängigkeit von dem Datum angezeigt. Es ist aber ebenso möglich, sich auf der Startseite Regeln zu einem bestimmten Datum anzeigen zu lassen ohne eine Anmeldung oder die Wahl eines bestimmten Produktes.

21.5 Nichtfunktionale Anforderungen

Die geforderten nicht-funktionalen Anforderungen Portabilität, Stabilität, Skalierbarkeit, Benutzerfreundlichkeit, Wartbarkeit und Sichtbarkeit sind in den verschiedenen Bereichen weitestgehend berücksichtigt worden.
Die Portabilität ist durch den durchgängigen Gebrauch von Java als Programmiersprache gesichert. Auch im Online-Shop wird Java in

Form von Java Server Pages verwendet. Durch die Behandlung möglicher Ausnahmen in Java (Exception Handling) wird gewährleistet, dass das System in einem stabilen Zustand verbleibt. Alle in den Testläufen aufgetretenen Fehler sind durch dieses Verfahren beseitigt worden. Das vorliegende System ist unabhängig von der Datenmenge funktionsfähig, jedoch können manche Analysen bzw. die Integration eine längere Zeitdauer in Anspruch nehmen. Die Bedienung des Systems ist möglichst einfach gehalten, insbesondere die Steuerung des Frameworks über Metadaten. Die Wartbarkeit des Systems wird vor allem durch die sehr ausführliche Dokumentation der Implementierungen in Javadoc realisiert. Der letzte Punkt, die Sicherheit, beschränkt sich, wie in der Anforderungsdefinition beschrieben, auf die Vergabe von Nutzerrechten in der Datenbank.

21.6 Benutzerschnittstellen und Fehlerverhalten

Wie im vorherigen Abschnitt erwähnt, sind die Benutzerschnittstellen möglichst einfach gestaltet worden. Die Integration ist hierbei über Parameter zu steuern, die Analysen über das Framework, welches Metadaten verwendet. Details der genauen Ausgestaltung der Schnittstellen sind den Ausführungen der Integration und der Analyse zu entnehmen. Das Fehlerverhalten ist bereits unter den nichtfunktionalen Anforderungen erwähnt worden. Die Behandlung aller aufgetretenen und erdenklichen Fehler dient der Stabilität des Systems und sorgt somit auch für eine Benutzerfreundlichkeit, da dieser Endbenutzer nicht programmiertechnische Fehlermeldungen selbst interpretieren muss.

21.7 Dokumentationsanforderungen

Der Anforderung entprechend ist für alle Schritte der Systementwicklung als auch der durchgeführten Analysen eine Dokumentation geschrieben worden. Die Dokumentation des vollständigen Sourcecodes erfolgte, wie beabsichtigt, in Form von Javadoc. Die angestrebte Unterteilung in ein Benutzer- und ein Developer-Handbuch ist nicht realisiert worden, stattdessen sind diese Bereiche in die einzelnen Dokumente als Unterabschnitte aufgenommen worden. Die gesamte Dokumentation erfolgte in LaTeX , wodurch die Arbeit auf unterschiedlichen Systemen erreicht werden konnte.

21.8 Zusammenfassung

Zusammenfassend ist festzustellen, dass fast alle Anforderungen vollständig umgesetzt worden sind. Anhand der Abbildung 21.1 werden die realisierten Komponenten des Gesamtszenarios aufgeführt. Der Verzicht auf einige andere wird erläutert.

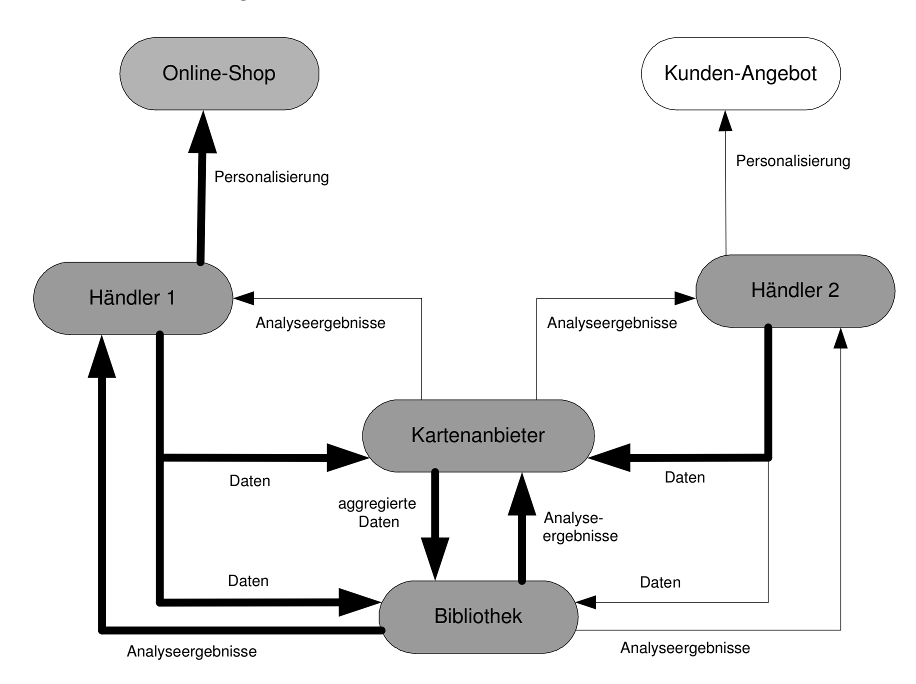


Abbildung 21.1: Übersicht Gesamtszenario

Die Datenbanken des Händler 1 und Händler 2 sowie des Kartenanbieters sind in einer früheren Phase realisiert worden und stehen somit im Gesamtszenario zur Verfügung. Die Bibliothek ist eine Erweiterung der WEKA-Bibliothek; insbesondere wurde ein Framework zur Steuerung dieser Bibliothek implementiert. Desweiteren ist ebenfalls ein Online-Shop entwickelt worden, der den Kunden von Händler 1 ein personalisiertes Angebot anbietet. Die Personalisierung für die Kunden von Händler 2 ist nicht weiter berücksichtigt worden, da nur der Online-Shop zur Evaluierung der Analyseergebnisse verwendet werden sollte (siehe Aufgabenstellung). In der Integration sind dennoch die Daten beider Händler in die Kartenanbieterdatenbank zusammengefügt worden. Auf diesen Daten sind anschließend Analysen mit

Hilfe der Bibliothek durchgeführt und die Ergebnisse in die Kartenanbieterdatenbank zurückgeschrieben worden. Diese Ergebnisse sind aufgrund ihrer Qualität nicht weiter verwendet worden und somit war ein Übertragen der Ergebnisse an die Händlerdatenbanken nicht notwendig. Die funktionierenden Analysen sind ausschließlich auf der Händler 1-Datenbank durchgeführt worden, wobei auch die Ergebnisse in dieser zurückgeschrieben worden sind. Der Händler 2 ist, wie oben beschrieben, nicht weiter betrachtet worden, so dass auch keine Analysen auf seiner Datenbank durchgeführt worden sind.
Die hier beschriebenen Einschränkungen haben sich durch die Aufgabenstellung bzw. durch die Verteilung der durchgeführten Analysen ergeben und bedeuten nicht, dass eine Anforderung an das System nicht erfüllt worden ist.

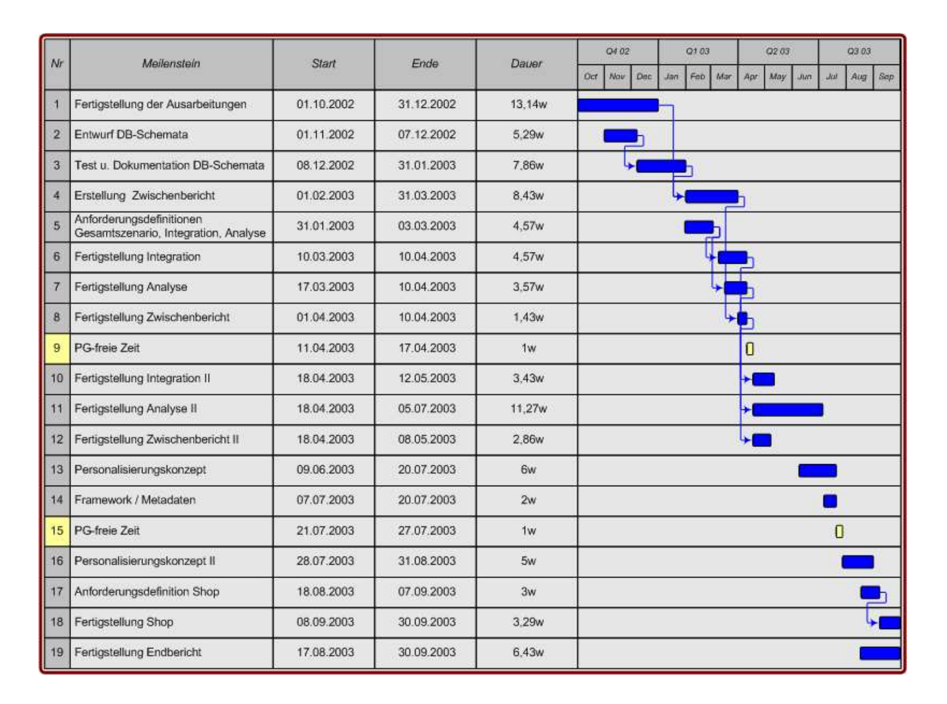
21.9 Meilensteine

Die folgende Darstellung (siehe Abbildung 21.2) zeigt die von den Projektgruppenmitgliedern festgelegten Meilensteine von Anfang bis Ende des Projekts. Wie schon im Zwischenbericht existiert zu jedem Meilenstein ein Start- und ein Enddatum. Aus diesen Daten ergibt sich die Bearbeitungsdauer eines Meilensteins in Wochen. Meilensteine, die miteinander durch eine Linie verbunden sind, werden als voneinander abhängige Meilensteine bezeichnet: der zeitlich vorangehende Meilenstein sollte vor Beginn des verbundenen Meilensteins abgeschlossen sein.
Vergleicht man den Meilensteinplan aus dem Zwischenbericht, so fällt auf, dass sich ab Meilenstein Nr. 5 Änderungen ergeben haben. Der alte Meilenstein Nr. 5, „Fertigstellung Zwischenbericht“ wird bis zum 08.05.03 verlängert, an diese Stelle sind dafür die Anforderungsdefinitionen getreten. Im Folgenden werden die einzelnen Meilensteine nun erläutert.

21.9.1 Fertigstellung Integration

Die Aufgaben, die sich aus der Anforderungsdefinition der Integration ergeben haben, sollen innerhalb dieses Meilensteins erledigt werden. Dieser und die folgenden Meilensteine werden durch insgesamt zwei Wochen projektfreier Zeit unterbrochen.

Meilenstein 6 und 10:



Nr	Meilenstein	Start	Ende	Dauer
1	Fertigstellung der Ausarbeitungen	01.10.2002	31.12.2002	13,14w
2	Entwurf DB-Schemata	01.11.2002	07.12.2002	5,29w
3	Test u. Dokumentation DB-Schemata	08.12.2002	31.01.2003	7,86w
4	Erstellung Zwischenbericht	01.02.2003	31.03.2003	8,43w
5	Anforderungsdefinitionen Gesamtszenario, Integration, Analyse	31.01.2003	03.03.2003	4,57w
6	Fertigstellung Integration	10.03.2003	10.04.2003	4,57w
7	Fertigstellung Analyse	17.03.2003	10.04.2003	3,57w
8	Fertigstellung Zwischenbericht	01.04.2003	10.04.2003	1,43w
9	PG-freie Zeit	11.04.2003	17.04.2003	1w
10	Fertigstellung Integration II	18.04.2003	12.05.2003	3,43w
11	Fertigstellung Analyse II	18.04.2003	05.07.2003	11,27w
12	Fertigstellung Zwischenbericht II	18.04.2003	08.05.2003	2,86w
13	Personalisierungskonzept	09.06.2003	20.07.2003	6w
14	Framework / Metadaten	07.07.2003	20.07.2003	2w
15	PG-freie Zeit	21.07.2003	27.07.2003	1w
16	Personalisierungskonzept II	28.07.2003	31.08.2003	5w
17	Anforderungsdefinition Shop	18.08.2003	07.09.2003	3w
18	Fertigstellung Shop	08.09.2003	30.09.2003	3,29w
19	Fertigstellung Endbericht	17.08.2003	30.09.2003	6,43w

Abbildung 21.2: Meilensteine Gesamtprojekt

10.03.03 bis *10.04.03* sowie *18.04.03* bis *12.05.03*, Dauer: 8,0 Wochen.

21.9.2 Fertigstellung Analyse

Parallel zur Fertigstellung der Integration beschäftigt sich eine Teilgruppe mit den Aufgaben der Analyse.

Meilenstein 7 und 11:
17.03.03 bis *10.04.03* sowie *18.04.03* bis *05.07.03*, Dauer: 14,84 Wochen.

21.9.3 Fertigstellung Zwischenbericht

Wie bereits oben erwähnt, wird der Meilenstein bis zum 08.05.03 verlängert.

Meilenstein 8 und 12:

01.04.03 bis *10.04.03* sowie *18.04.03* bis *08.05.03*, Dauer: 4,29 Wochen.

21.9.4 PG-freie Zeit

Gemäß der Projektgruppenordnung werden insgesamt zwei Wochen eingeplant in denen die Arbeit ruht. Die Projektgruppe hat sich entschlossen, diese zwei Wochen in jeweils eine Woche aufzuteilen.

Meilenstein 9 und 15:
11.04.03 bis *17.04.03* sowie *21.07.03* bis *27.07.03*, Dauer: 2,0 Wochen.

21.9.5 Personalisierungskonzept

Es soll ein Konzept erstellt werden, dass Vorschläge zur Personalisierung des Shops und zur konkreten Umsetzung der Analyseergebnisse macht. Dazu gehört auch die Anwendung der Analysealgorithmen, die durch die Analysegruppe bereitgestellt werden.

Meilenstein 13 und 16:
09.06.03 bis *20.07.03* sowie *28.07.03* bis *31.08.03*, Dauer: 11,0 Wochen.

21.9.6 Framework / Metadaten

Die Implementierung des Frameworks und die Unterstützung von Metadaten wird als zusätzlicher Meilenstein losgelöst vom eigentlichen Analyse-Meilenstein geplant.

Meilenstein 14:
07.07.03 bis *20.07.03*, Dauer: 2,0 Wochen.

21.9.7 Anforderungsdefinition Shop

Diese Aufgabe, die eng mit dem Personalisierungskonzept verbunden ist, soll die Umsetzung der Analyseergebnisse aus der Sicht der Shops erarbeiten.

Meilenstein 17:
18.08.03 bis *07.09.03*, Dauer: 3,0 Wochen.

21.9.8 Fertigstellung Shop

Implementierung und Anwendung der Analyseergebnisse, so dass sie im Shop präsentiert werden, soll innerhalb dieses Meilensteins geschehen.

Meilenstein 18:
08.09.03 bis *30.09.03*, Dauer: 3,29 Wochen.

21.9.9 Fertigstellung Endbericht

Als letzten Meilenstein gilt es den Endbericht anzufertigen. Obwohl die Erfahrung mit dem Zwischenbericht gezeigt hat, dass aufgrund von Korrekturdurchläufen etc. die tatsächlich benötigte Zeit zum Abschließen des Berichts schlecht planbar ist, endet dieser Meilenstein mit dem offiziellen Ende des Projekts.

Meilenstein 18:
17.08.03 bis *30.09.03*, Dauer: 6,43 Wochen.

21.10 Meilensteinfazit

Die Meilensteinplanung in Abbildung 21.2 stellt die Sollplanung dar. Tatsächlich gibt es Abweichungen vom ursprünglichen Zeitplan. In manchen Fällen, z.B. Implementierung des Shops, konnten die Arbeiten früher als geplant, in anderen, z.B. das Personalisierungskonzept, später abgeschlossen werden. Insgesamt läßt sich sagen, dass das Projekt alle aufgetragenen und im Meilensteinplan aufgeführten Aufgaben bis zum Ende des Projekts bewältigt hat. Details zu den einzelnen Meilensteinen ist den jeweiligen Kapiteln der Teilgruppen zu entnehmen.

22 Zusammenfassung und Ausblick

Im Folgenden sollen die Ergebnisse der Projektgruppe noch einmal zusammenfassend betrachtet werden. Insbesondere soll dabei ein Ausblick gegeben werden, wie die Ergebnisse in Zukunft verwendet werden können und wo sich Erweiterungen oder Verbesserungen anbringen ließen.
Dieses Kapitel wird so gegliedert, dass jede Teilaufgabe einzeln betrachtet wird und abschließend eine Schlussbetrachtung für die gesamte Projektgruppe stattfindet.

22.1 Gesamtszenario

Die erste Aufgabe innerhalb der Projektgruppe war der Entwurf eines geeigneten Gesamtszenarios. Da einer der Schwerpunkte der Projektgruppe die Verteiltheit der Analysen war, wurde das Gesamtszenario so gewählt, dass zwei Händler und ein Kartenanbieter existieren (siehe Kapitel 2). Dadurch sollte vor allem gezeigt werden, dass das zu entwerfende Framework an beliebiger Stelle eingesetzt werden kann, und auch, das Analyseergebnisse ausgetauscht werden können.
Im Verlauf der Projektgruppe hat sich jedoch herausgestellt, dass das sehr groß gewählte Gesamtszenario nicht komplett abgedeckt werden konnte. Im Speziellen hat die nötige Integration relativ viel Zeit benötigt, die unter Umständen besser in die Entwicklung eines umfangreicheren Frameworks oder Personalisierungskonzeptes hätte investiert werden können. Es lässt sich feststellen, dass die Verteiltheit des Frameworks zwar gegeben ist, aber das Personalisierungskonzept sowie die im Online-Shop verwendeten Analysen keinen Gebrauch von der Verteiltheit machen und vor allem nur auf der Datenbank des Händler 1 arbeiten. Da die Integration ohnehin nur rudimentär umgesetzt wurde, kann abschließend gesagt werden, dass ein Szenario mit nur einem Händler für die Zwecke der Projektgruppe ausgereicht hätte.

22.2 Integration

Die Aufgabe der Integration hat den Import aller Daten aus den Händlerdatenbanken in die Datenbank des Kartenanbieters umfasst. Dieses ist nötig gewesen, damit der Kartenanbieter seine Analysen auf den Daten der Händler durchführen konnte. Die genaue Durchführung und die dabei entstandenen Probleme sind im Teil II zur Dokumentation der Integration nachzulesen.
Der gewählte Lösungsweg hat sich auf Anraten der Projektgruppenleiter meist auf die Implementation mit dem geringsten Zeitaufwand beschränkt, da der Fokus der zu leistenden Arbeit auf der Erstellung der Personalisierungsbibliothek gelegen und die Integration hierfür nur einen notwendigen Hilfsschritt dargestellt hat, und nicht Ziel der Projektgruppe gewesen ist.
Eine Aggregation der Datensätze auf eine Ebene oder das Zusammenfassen von mehreren Datensätzen mit der selben ID zu einem anstelle der Löschung dieser fälschlicherweise doppelt vorhandenen Datensätze wäre eine alternative Lösung gewesen, die jedoch aus o.g. Gründen nicht umgesetzt worden ist.
Ebenso ist aus zeitlichen Gründen auf eine grafische Benutzeroberfläche verzichtet worden. Anstelle dieser ist eine parametrische Steuerung implementiert worden, die zeitlich weniger Entwicklungsaufwand dargestellt hat.
Bezüglich der Erweiterbarkeit des Integrationsszenarios sei hier auf den Kapitel 7 verwiesen.

22.3 Analyse

Aufgabe der Analyse-Gruppe war die Konzeption und Umsetzung des verteilten Analyse- und Personalisierungs-Frameworks. Diese Aufgabe wurde von der Gruppe in drei Teile aufgeteilt (siehe Kapitel 10).
Die Umsetzung des Caches für WEKA wurde wie gewünscht durchgeführt. Wie erwartet wurden dadurch Analysen mit WEKA bei gleichbleibendem Hauptspeicherverbrauch möglich. Wie in der Dokumentation zum Cache erwähnt (siehe Kapitel 12), gibt es jedoch mit einigen Algorithmen noch Probleme. Da es nie eine zentrale Aufgabe der Projektgruppe war, einen Cache für WEKA zu schreiben, wurde an dieser Stelle bewusst auf die Umsetzung komplexerer Caching-Algorithmen verzichtet. Die von der Projektgruppe verwendeten Al-

gorithmen konnten problemlos ausgeführt werden und daher wurde die aktuelle Implementierung als ausreichend angesehen. Vorschläge zur Erweiterung des Caches sind in der oben genannten Dokumentation unter Abschnitt 12.3 zu finden.
Die zweite Teilaufgabe war die Implementierung eines Algorithmus zum Auffinden kalendarischer Muster (siehe Kapitel 13). Dieser Algorithmus wurde komplett inklusive der Unterstützung des Fuzzy-Match implementiert. Zum Testen des Algorithmus wurden mit Hilfe einer Java-Klasse (*tBasket* [LNWJ]) Testdaten erzeugt, die entsprechende Muster enthalten. Testläufe des Algorithmus haben gezeigt, dass er in der Lage ist diese Muster zu finden. Insbesondere werden die Ergebnisse des Algorithmus auch im Online-Shop verwendet (siehe Abschnitt 22.5). Wie in Kapitel 13 beschrieben, wäre die Unterstützung anderer kalendarischer Muster eine Erweiterungsmöglichkeit für den Algorithmus. Während aktuell nur Muster der Form `<Jahr, Monat, Tag>` gefunden werden, wäre es denkbar auch Muster anderer Granularität bzw. Zusammensetzung zu suchen.
Die dritte Teilaufgabe bestand darin, ein durch Metadaten steuerbares Framework zu entwickeln, welches die für das Personalisierungskonzept notwendigen Analysen in einem verteilten Szenario ausführen kann. Dabei ist, wie in Kapitel 14 beschrieben, eine Bibliothek entstanden, die die genannten Anforderungen (siehe Kapitel 10 und Abschnitt 14.1) erfüllt (siehe Abschnitt 14.6). Das Framework leistet sogar mehr, als gefordert.
Der Nachteil des Framework ist die Verwendung eines eigenen anstelle eines standardisierten Metadatenformats. Es wurde ein eigenes Format verwendet, da die Einarbeitung in ein bestehendes Format, wie etwa CWM [Obj01], als zu zeitintensiv angesehen wurde. Die Anpassung an einen Metadatenstandard wäre sicherlich eine sinnvolle Erweiterung.

22.4 Personalisierungskonzept

Aufgabe des Personalisierungskonzeptes war die Entwicklung von Konzepten zur Personalisierung eines Online-Shops. Insbesondere sollten die Ergebnisse dieser Gruppe im Online-Shop der Projektgruppe als Grundlage der Personalisierung dienen.
Im Rahmen des Personalisierungskonzeptes wurden diverse theoretische Verfahren zu Analyse von Kunden- sowie Transaktionsdaten,

zum Teil auch in Kombination miteinander, entwickelt (siehe Kapitel 15 und Kapitel 17). Einige dieser Ideen wurden auf Basis der Testdaten des Kartenanbieters bzw. des Online-Shops praktisch erprobt (siehe Kapitel 16 und Abschnitt 18).
Im Online-Shop wurden letztlich aus zeitlichen Gründen nur relativ einfache Analysen durchgeführt. Zum einen wurden zwei Top5-Listen anhand des demographischen Clusterings erstellt. Zum anderen werden ebenfalls die Ergebnisse des kalendarischen Algorithmus im Online-Shop präsentiert.
Aufgrund mangelnder Zeit konnten am Ende leider nicht mehr die kombinierten Analyseverfahren umgesetzt werden. Sicherlich wäre es interessant gewesen, die Ergebnisse im Online-Shop direkt vergleichen zu können und so zu verifizieren, ob die eher theoretischen Vorüberlegungen in der „Realität“ gelten.

22.5 Online-Shop

Der Online-Shop ist implementiert worden, um die Ergebnisse der Personalisierungsbibliothek, die im Rahmen der Projektgruppe entwickelt worden ist und das Kern-Element der Projektgruppenarbeit darstellt, zu visualisieren. Hierbei ist vor allem darauf geachtet worden, dass die Funktionalität und nicht das optische Design im Vordergrund steht.
So sind nur die notwendigsten Funktionalitäten wie eine Katalogübersicht, eine Produktdetailseite, ein Warenkorb und ein rudimentäres User-Management implementiert worden. Im Rahmen der Projektgruppe erscheinen diese Einschränkungen sinnvoll, im Rahmen eines Online-Shops könnten noch viele Features wie z.B. Produktbilder oder eine komplexere Benutzerführung umgesetzt werden.
Die Darstellung der fünf meistgekauften Produkte der Kunden eines Clusters ist in der vorliegenden Implementierung ausgelegt, die Ergebnisse der Personalisierungsanalysen direkt zu visualisieren. Denkbar wäre hier, auf Basis der Informationen über den angemeldeten Kunden und die jeweiligen fünf meistverkauften Produkte der beiden Cluster, zu denen ein Kunde zugeordnet worden ist, eine einzelne Produktempfehlungsliste umzusetzen, die nicht einfach nur alle gefundenen Produkte untereinander darstellt, sondern diese miteinander vergleicht und nur eine Liste mit allen voneinander unterschiedlichen Produkten darstellt.

Teil VI

Anhang

Integration

Die im Folgenden dargestellten Tabellen dienten als Grundlage für die Arbeit der Integration. Sie beinhalten alle Attribute und deren Wertebereiche der beiden Händlerdatenbanken. Diese werden den entsprechenden Attributen der Kartenanbieterdatenbank gegenübergestellt, wobei für einzelne Attribute im Kartenanbieter-Schema erst noch Entsprechungen generiert werden mussten, da diese nicht vorhanden waren.

kunde_ohne_karte (H2)	Wertebereich	Cardprovider	Wertebereich
transaktions_id	NUMBER(38) NOT NULL	egal	-
plz	NUMBER(38)	Kunden_Adresse(PLZ)	VARCHAR2(40)
kg_id	CHAR(18) NOT NULL	generieren?	

Tabelle A.1: Relation kunde_ohne_karte

filiale (H2)	Wertebereich	Cardprovider	Wertebereich
adr_str adr_plz adr_ort	VARCHAR2(20) NOT NULL CHAR(18) CHAR(18)	Shop(Strasse) Shop(PLZ) Shop(Ort)	VARCHAR2(40) NOT NULL
filial_id vid_filiale	NUMBER(38) NOT NULL	Shop(ShopID)	NUMBER(38) NOT NULL
gza_filiale	DATE NOT NULL	bearbeiten	?
gze_filiale	DATE	bearbeiten	?
tza_filiale	DATE NOT NULL	bearbeiten	?
tze_filiale	DATE	bearbeiten	?

Tabelle A.2: Relation Filiale

Kategorie (H2)	Wertebereich	Cardprovider	Wertebereich
kategorie_id	NUMBER(38) NOT NULL	Warengruppe(WarengruppeID)	NUMBER(38) NOT NULL
artikelklasse	VARCHAR2(20)	Warengruppe(Name)	VARCHAR2(40)

Tabelle A.3: Relation Kategorie

kunde_mit_karte (h2)	Wertebereich	Cardprovider	Wertebereich
kunden_id vid_kk	NUMBER(38) NOT NULL	Kunden(KundenID)	INTEGER NOT NULL
anrede	VARCHAR2(20) NOT NULL	Kunden(Geschlecht)	VARCHAR2(40)
vorname name	VARCHAR2(20) NOT NULL	Kunden_Name(Vorname) Kunden_Name(Name)	VARCHAR2(40)
geb_datum	DATE NOT NULL	Kunden(Geburtsdatum)	DATE
adr_str adr_plz adr_ort	VARCHAR2(30) NUMBER(38) VARCHAR2(20)	Kunden_Adresse(Strasse) Kunden_Adresse(Ort) Kunden_Adresse(PLZ)	VARCHAR2(40)
anz_kinder	NUMBER(38)	generieren	?
anz_personen	NUMBER(38)	generieren	?
kunde_alter	NUMBER(38)	egal	-
telefonnummer	NUMBER(20)	generieren	?
email	VARCHAR2(50) NOT NULL	generieren	?
fam_id	NUMBER(38) NOT NULL (LoV)	Kunden_Familienstand(Familienstand)	NUMBER(38) NOT NULL (lov)
kontakt_id	NUMBER(38) NOT NULL (LoV)	generieren	?
kennt_id	NUMBER(38) NOT NULL (LoV)	generieren	?
beruf_id	NUMBER(38) NOT NULL (LoV)	Kunden_Beruf(Beruf)	VARCHAR2(40) NOT NULL
hobby_id	NUMBER(38) NOT NULL (LoV)	generieren	?
abschluss_id	NUMBER(38) NOT NULL (LoV)	generieren	?
gza_kk	DATE NOT NULL	bearbeiten	?
gze_kk	DATE	bearbeiten	?
tza_kk	DATE NOT NULL	bearbeiten	?
tze_kk	DATE	bearbeiten	?

Tabelle A.4: Relation kunde_mit_karte

preis (H2)	Wertebereich	Cardprovider	Wertebereich
geldwert	NUMBER(38) NOT NULL	Sortiment(VK)	NUMBER(38)
preis_id vid_preis	NUMBER(38) NOT NULL	egal	egal
gza_preis	DATE NOT NULL	Sortiment(gza)	DATE
gze_preis	DATE	Sortiment(gze)	DATE
tza_preis	DATE NOT NULL	Sortiment(tza)	DATE NOT NULL
tze_preis	DATE	Sortiment(tze)	DATE

Tabelle A.5: Relation Preis

artikel (H2)	Wertebereich	Cardprovider	Wertebereich
artikelnummer vid_artikel	NUMBER(38) NOT NULL	Produkt(ProduktID)	NUMBER(38) NOT NULL
verfuegbar_id	NUMBER(38) NOT NULL	generieren	?
bezeichnung	VARCHAR2(20)	Produkt(Name)	VARCHAR2(40)
hersteller	VARCHAR2(20)	Produkt(Hersteller)	VARCHAR2(40)
kategorie_id	NUMBER(38) NOT NULL (LoV)	Einordnung(WarengruppeID)	NUMBER(38) NOT NULL
preis_id vid_preis	NUMBER(38) NOT NULL	egal	egal
gza_preis	DATE NOT NULL	bearbeiten	?
gze_preis	DATE	bearbeiten	?
tza_preis	DATE NOT NULL	bearbeiten	?
tze_preis	DATE	bearbeiten	?

Tabelle A.6: Relation Artikel

position (H2)	Wertebereich	Cardprovider	Wertebereich
artikelnummer vid_artikel	NUMBER(38) NOT NULL	Posten(ProduktID)	NUMBER(38) NOT NULL
anzahl	NUMBER(38) NOT NULL	Posten(Anzahl)	NUMBER(38)
bon_id	NUMBER(38) NOT NULL	Posten(TransaktionsID)	NUMBER NOT NULL
positions_id	NUMBER(38) NOT NULL	egal	egal

Tabelle A.7: Relation Position

bon (H2)	Wertebereich	Cardprovider	Wertebereich
bon_id	NUMBER(38) NOT NULL	Transaktion(TransaktionID)	NUMBER NOT NULL
filial_id vid_filiale	NUMBER(38) NOT NULL	Transktion(ShopID)	NUMBER(38) NOT NULL
kunden_id vid_kk	NUMBER(38)	Rechnung(KundenID)	NUMBER(38) NOT NULL
transaktions_id	NUMBER(38)	Rechnung(KundenID)	NUMBER(38) NOT NULL
uhrzeit	DATE NOT NULL	Transaktion(Zeitpunkt)	DATE NOT NULL

Tabelle A.8: Relation Bon

Kunde (H1)	Wertebereich	Cardprovider	Wertebereich
knd_id, knd_sid	NUMBER(38) NOT NULL NUMBER(38) NOT NULL	Kunden(KundenID)	NUMBER(38) NOT NULL
name vorname	VARCHAR2(20)	Kunden_Name(Name) Kunden_Name(Vorname)	VARCHAR2(40) VARCHAR2(40)
gebdatum	DATE	Kunden(Geburtsdatum)	DATE
geschlecht	NUMBER(38)	Kunden(Geschlecht)	VARCHAR2(40)
email	VARCHAR2(255)	generieren	-
kennwort	VARCHAR2(20)	egal	egal
gza	DATE NOT NULL	bearbeiten	?
gze	DATE	bearbeiten	?
tza	DATE NOT NULL	bearbeiten	?
tze	DATE	bearbeiten	?

Tabelle A.9: Relation Kunde

Adresse (H1)	Wertebereich	Cardprovider	Wertebereich
name vorname	VARCHAR2(20) VARCHAR2(20)	Kunden_Name(Name) Kunden_Name(Vorname)	VARCHAR2(40) VARCHAR2(40)
strasse, hausnummer, plz, ort, land	VARCHAR2(20)	Kunden_Adresse(Strasse, Hausnummer, PLZ, Ort, Staat)	VARCHAR2(40)
telefon, telefax	VARCHAR2(20)	generieren	
gza	DATE NOT NULL	bearbeiten	?
gze	DATE	bearbeiten	?
tza	DATE NOT NULL	bearbeiten	?
tze	DATE	bearbeiten	?

Tabelle A.10: Relation Adresse

Profil (H1)	Wertebereich	Cardprovider	Wertebereich
typ name beschreibung	NUMBER(38) VARCHAR2(20) VARCHAR2(255)	generieren?? Analyseaufgabe??	?
gza	DATE NOT NULL	bearbeiten	?
gze	DATE	bearbeiten	?
tza	DATE NOT NULL	bearbeiten	?
tze	DATE	bearbeiten	?

Tabelle A.11: Relation Profil

Produkt (H1)	Wertebereich	Cardprovider	Wertebereich
p_id, p_sid	NUMBER(38) NOT NULL	Produkt(ProduktID)	NUMBER(38) NOT NULL
name	VARCHAR2(20)	Produkt(Name)	VARCHAR2(40)
beschreibung	VARCHAR2(255)	Produkt(Beschreibung)	VARCHAR2(255)
e_preis	FLOAT(10)	Sortiment(EK)	NUMBER(38)
v_preis	FLOAT(10)	Sortiment(VK)	NUMBER(38)
versand_aufschlag	NUMBER(38)	Transaktion(Kosten)	NUMBER(38)
rabattfaehig	NUMBER(38)	bearbeiten	bearbeiten
prioritaet	NUMBER(38)	generieren	-
status	NUMBER(38)	generieren (siehe H2)	-
pt_id pt_sid	NUMBER(38) NUMBER(38)	egal	egal
h_id h_sid	NUMBER(38)	Produkt(Hersteller)	VARCHAR2(40)
gza	DATE NOT NULL	bearbeiten	?
gze	DATE	bearbeiten	?
tza	DATE NOT NULL	bearbeiten	?
tze	DATE	bearbeiten	?

Tabelle A.12: Relation Produkt

Kategorie (H1)	Wertebereich	Cardprovider	Wertebereich
parent_id parent_sid	NUMBER(38)	Warengruppe(ObergruppeID)	NUMBER(38)
name	VARCHAR2(20)	Warengruppe(Name)	VARCHAR2(40)
beschreibung	VARCHAR2(255)	Warengruppe(Beschreibung)	VARCHAR2(255)
k_id k_sid	NUMBER(38) NOT NULL NUMBER(38) NOT NULL	Warengruppe(WarengruppeID) Warengruppe(tza_Warengruppe)	NUMBER(38) NOT NULL DATE NOT NULL
gza	DATE NOT NULL	bearbeiten	?
gze	DATE	bearbeiten	?
tza	DATE NOT NULL	bearbeiten	?
tze	DATE	bearbeiten	?

Tabelle A.13: Relation Kategorie

Hersteller (H1)	Wertebereich	Cardprovider	Wertebereich
h_id h_sid	NUMBER(38) NOT NULL	Produkt(ProduktID) Produkt(tza_Produkt)	NUMBER(38) NOT NULL DATE NOT NULL
name	VARCHAR2(100)	Produkt(Hersteller)	VARCHAR2(40)
ap_name ap_vorname ap_geschlecht	VARCHAR2(20)	egal	egal
email	VARCHAR2(20)	egal	egal
strasse hausnummer plz ort land	VARCHAR2(50) VARCHAR2(20) VARCHAR2(20) VARCHAR2(50) VARCHAR2(50)	egal	egal
telefon, telefax, url	VARCHAR2(20)	egal	egal
tza	date	bearbeiten	-
tze	date	bearbeiten	-
gza	date	bearbeiten	-
gze	date	bearbeiten	-

Tabelle A.14: Relation Hersteller

Produkt_Warenkorb (H1)	Wertebereich	Cardprovider	Wertebereich
p_id p_sid	NUMBER(38) NOT NULL	Posten(ProduktID)	NUMBER(38) NOT NULL
wk_id wk_sid	NUMBER(38)	Posten(TransaktionsID)	NUMBER NOT NULL
pw_sid	NUMBER(38) NOT NULL	egal	egal
menge	NUMBER(38)	Posten(Anzahl)	NUMBER(38)
tza	date	bearbeiten	-
tze	date	bearbeiten	-
gza	date	bearbeiten	-
gze	date	bearbeiten	-

Tabelle A.15: Relation Produkt Warenkorb

Warenkorb (H1)	Wertebereich	Cardprovider	Wertebereich
wk_id wk_sid	NUMBER(38) NOT NULL	Transaktion(TransaktionID)	NUMBER NOT NULL
k_id knd_sid	NUMBER(38)	Rechnung(KundenID)	NUMBER(38) NOT NULL
tza	date	bearbeiten	-
tze	date	bearbeiten	-
gza	date	bearbeiten	-
gze	date	bearbeiten	-

Tabelle A.16: Relation Warenkorb

Attribut (H1)	Wertebereich	Cardprovider	Wertebereich
a_id a_sid	NUMBER(38) NOT NULL	ProduktAttribut(ProduktAttributID)	NUMBER(38) NOT NULL
name	VARCHAR2(20)	ProduktAttribut(Name)	VARCHAR2(40)
beschreibung	VARCHAR2(255)	ProduktAttribut(Beschreibung)	VARCHAR2(255)
tza	date	bearbeiten	-
tze	date	bearbeiten	-
gza	date	bearbeiten	-
gze	date	bearbeiten	-

Tabelle A.17: Relation Attribut

Text_Attribut (H1)	Wertebereich	Cardprovider	Wertebereich
p_id p_sid	NUMBER(38) NOT NULL	Textattribut(ProduktID) Textattribut(tza_Produkt)	NUMBER(38) NOT NULL DATE NOT NULL
a_id a_sid	NUMBER(38) NOT NULL	Textattribut(ProduktAttributID)	NUMBER(38) NOT NULL
ta_sid	NUMBER(38) NOT NULL	egal	egal
wert	VARCHAR2(20)	TextAttribut(Wert)	BLOB
tza	date	bearbeiten	-
tze	date	bearbeiten	-
gza	date	bearbeiten	-
gze	date	bearbeiten	-

Tabelle A.18: Relation Text Attribut

Number_Attribut (H1)	Wertebereich	Cardprovider	Wertebereich
p_id p_sid	NUMBER(38) NOT NULL	NumberAttribut(ProduktID) NumberAttribut(tza_Produkt)	NUMBER(38) NOT NULL DATE NOT NULL
a_id a_sid	NUMBER(38) NOT NULL	NumberAttribut(ProduktAttributID)	NUMBER(38) NOT NULL
na_sid	NUMBER(38) NOT NULL	egal	egal
wert	NUMBER(38)	NumberAttribut(Wert)	NUMBER(38)
tza	date	bearbeiten	-
tze	date	bearbeiten	-
gza	date	bearbeiten	-
gze	date	bearbeiten	-

Tabelle A.19: Relation Number Attribut

String_Attribut (H1)	Wertebereich	Cardprovider	Wertebereich
p_id p_sid	NUMBER(38) NOT NULL	StringAttribut(ProduktID) StringAttribut(tza_Produkt)	NUMBER(38) NOT NULL DATE NOT NULL
a_id a_sid	NUMBER(38) NOT NULL	StringAttribut(ProduktAttributID)	NUMBER(38) NOT NULL
sa_sid	NUMBER(38) NOT NULL	egal	egal
wert	VARCHAR2(255)	StringAttribut(Wert)	VARCHAR2(40)
tza	date	bearbeiten	-
tze	date	bearbeiten	-
gza	date	bearbeiten	-
gze	date	bearbeiten	-

Tabelle A.20: Relation String Attribut

Zahlung (H1)	Wertebereich	Cardprovider	Wertebereich
za_id za_sid	NUMBER(38) NOT NULL	Transaktion(TransaktionID)	NUMBER NOT NULL
b_id b_sid	NUMBER(38) NOT NULL	egal	-
z_sid	NOT NULL	bearbeiten	-
typ	NUMBER(38) NOT NULL	Transaktion(Art)	NUMBER(38) NOT NULL (LoV)
betrag	NUMBER(38)	Rechnung(Betrag)	NUMBER(38)
tza	date	bearbeiten	-
tze	date	bearbeiten	-
gza	date	bearbeiten	-
gze	date	bearbeiten	-

Tabelle A.21: Relation Attribut

Personalisierung

Nicht-temporale Analysen

Übersicht der Konfigurationsdatei, mit der aus dem Paket *diko.framework* der *AnalysisRunner* aufgerufen wird. **configKIDGebGes:**
source= diko.framework.input.ConfigurableJDBCSourceIterator
relation= foo
source.jdbcDriver=oracle.jdbc.driver.OracleDriver
source.jdbcConnection=jdbc:oracle:thin:
@power2.offis.uni-oldenburg.de:1521:power2
source.jdbcLogin= xxxx
source.jdbcPassword= xxxx

source.jdbcSQL= select distinct knd_id, geschlecht, to_char(gebdatum, 'YYYY') from kunde
attributeName_0= kundenid
attributeType_0= nominal
attributeName_1= geschlecht
attributeType_1= numeric
attributeName_2= geburtsdatum
attributeType_2= numeric

sink=diko.framework.output.JDBCWriter
sink.jdbcDriver= oracle.jdbc.driver.OracleDriver
sink.jdbcConnection=jdbc:oracle:thin:
@power2.offis.uni-oldenburg.de:1521:power2
sink.jdbcLogin = xxxx
sink.jdbcPassword = xxxx
sink.jdbcTable = clustKundenGebGes

algorithm= diko.framework.algorithms.SimpleKMeansController
kMeans.numClusters=10
kMeans.seed=-30
clusterer.FilterColumns=1

anarunnergebges.cfg:
analysis_0 = configKIDGebGes
analysis_0.postprocessor_0= diko.personalisierung.
kundenattributte.PostprocessorGebGes

configKidGebPlz:
source= diko.framework.input.ConfigurableJDBCSourceIterator
relation= foo
source.jdbcDriver=oracle.jdbc.driver.OracleDriver
source.jdbcConnection=jdbc:oracle:thin:
@power2.offis.uni-oldenburg.de:1521:power2
source.jdbcLogin= xxxx
source.jdbcPassword= xxxx

source.jdbcSQL= select k.knd_id, to_char(k.gebdatum, 'YYYY'), a.plz from adresse a, kunde k, adresse_kunde ak where k.knd_id=ak.knd_id and k.knd_sid=ak.knd_sid and ak.a_id=a.a_id and ak.a_sid=a.a_sid and ak.typ=1
attributeName_0= kundenid
attributeType_0= nominal
attributeName_1= geburtsdatum
attributeType_1= numeric
attributeName_2= plz
attributeType_2= numeric

sink=diko.framework.output.JDBCWriter
sink.jdbcDriver= oracle.jdbc.driver.OracleDriver
sink.jdbcConnection=jdbc:oracle:thin:
@power2.offis.uni-oldenburg.de:1521:power2
sink.jdbcLogin = xxxx
sink.jdbcPassword = xxxx
sink.jdbcTable = clustKundenGebPlz

algorithm= diko.framework.algorithms.SimpleKMeansController
kMeans.numClusters=9
kMeans.seed=5
clusterer.FilterColumns=1

anarunnergebplz.cfg:

analysis_0 = configKidGebPlz
analysis_0.postprocessor_0=
diko.personalisierung.kundenattributte.PostprocessorGebPlz

Clusteranalyse nach Kunden und gekauften Produkten

In diesem Abschnitt sind die Ergebnisse der Analysen mit dem Algorithmus SimpleKMeans aufgelistet, wie sie von der WEKA-GUI ausgegeben werden und die als Grundlage der Bewertungen des Verfahrens bzw. der Daten dienten. Zu Beginn eines jeden Ergebnisses sind die Parameter aufgelistet: N gibt den Wert der maximalen Clusteranzahl an, S den Wert für den seed. Weiterhin sind die Werte der jeweiligen Centroide der gefundenen Cluster sowie die Verteilung der Kunden auf diese (absolut und relativ) dargestellt.
Anhand der Ausgabe der ersten Analyse werden die Zusammenhänge genauer erläutert: Die Analyse wird mit dem Wert 5 für die Clusteranzahl und einem seed von -2 durchgeführt. Anschließend sind die fünf Centroiden der Cluster mit den Werten für die sieben hier betrachteten Attribute (*WG1* bis *WG7*) aufgeführt. Das Cluster mit der Nummer 2 hat nur für *WG5* einen Wert ungleich null, d.h. in dieses Cluster fallen alle Kunden, die nur Produkte der Warengruppe 5 erworben haben. Die sieben Nullwerte in für den Centroiden von Cluster 2 symbolisiert die Gruppe der Nichtkäufer. Zum Abschluss einer jeden Analyse wird die Verteilung der Kunden aufgezeigt, wobei ein Eintrag (*2 324 (32%*) bedeutet, dass in Cluster 2 324 Kunden eingeteilt wurden, was 32 Prozent der gesamten Kunden entspricht.

Scheme: weka.clusterers.SimpleKMeans -N 5 -S -2
Cluster centroids:
Cluster 0 14.214285714285714 12.142857142857142 13.147619047619047 17.285714285714285 12.957142857142857 13.719047619047618 12.976190476190476
Cluster 1 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 2 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 3 16.88888888888889 12.793650793650794 13.551587301587302 13.05952380952381 13.035714285714286 13.984126984126984 13.234126984126984

Cluster 4 12.184331797235023 13.90783410138249 14.184331797235023 13.2073732718894 13.76958525345622 15.064516129032258 14.013824884792626

0 210 (21%) 1 1 (0%) 2
324 (32%) 3 252 (25%) 4 217 (22%)

Scheme: weka.clusterers.SimpleKMeans -N 5 -S -1
Cluster centroids:
Cluster 0 16.544159544159545 12.49002849002849 13.452991452991453 14.227920227920228 12.877492877492877 13.897435897435898 13.042735042735043
Cluster 1 12.432926829268293 13.439024390243903 13.817073170731707 14.612804878048781 13.640243902439025 14.621951219512194 13.789634146341463
Cluster 2 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 3 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 4 0.0 0.0 0.0 0.0 0.0 0.0 0.0

0 351 (35%) 1 328 (33%) 2
1 (0%) 3 324 (32%)

Scheme: weka.clusterers.SimpleKMeans -N 5 -S 0
Cluster centroids:
Cluster 0 0.0 0.0 0.0 0.0 0.3076923076923077 0.0 0.0
Cluster 1 12.60135135135135 12.844594594594595 13.283783783783784 16.9527027027027 13.385135135135135 13.777027027027026 13.472972972972974
Cluster 2 16.862068965517242 12.985221674876847 13.704433497536947 12.47783251231527 13.216748768472906 14.014778325123153 13.25615763546798
Cluster 3 16.1156462585034 11.775510204081632 13.122448979591837 16.6734693877551 12.435374149659864 13.714285714285714 12.714285714285714
Cluster 4 12.30939226519337 13.94475138121547 14.237569060773481 12.67403314917127 13.823204419889503 15.32596685082873 14.07182320441989

0 325 (32%) 1 148 (15%) 2
203 (20%) 3 147 (15%) 4 181 (18%)

Scheme: weka.clusterers.SimpleKMeans -N 5 -S 1
Cluster centroids:
Cluster 0 16.88888888888889 12.793650793650794 13.551587301587302 13.05952380952381 13.035714285714286 13.984126984126984 13.234126984126984
Cluster 1 14.214285714285714 12.142857142857142 13.147619047619047 17.285714285714285 12.957142857142857 13.719047619047618 12.976190476190476
Cluster 2 12.184331797235023 13.90783410138249 14.184331797235023 13.2073732718894 13.76958525345622 15.064516129032258 14.013824884792626
Cluster 3 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 4 0.0 0.0 0.0 0.0 0.0 0.0 0.0

0 252 (25%) 1 210 (21%) 2
217 (22%) 31 (0%) 4 324 (32%)

Scheme: weka.clusterers.SimpleKMeans -N 5 -S 2
Cluster centroids:
Cluster 0 18.042857142857144 12.507142857142858 13.371428571428572 13.207142857142857 12.878571428571428 13.05 13.542857142857143
Cluster 1 0.0 0.0 0.0 0.0 0.3076923076923077 0.0 0.0
Cluster 2 14.154696132596685 12.060773480662984 12.988950276243093 17.56353591160221 13.005524861878452 13.657458563535911 13.011049723756907
Cluster 3 11.297101449275363 14.246376811594203 14.442028985507246 13.304347826086957 13.76086956521739 14.44927536231884 14.782608695652174
Cluster 4 14.718181818181819 13.145454545454545 13.809090909090909 13.286363636363637 13.354545454545455 15.368181818181819 12.772727272727273

0 140 (14%) 1 325 (32%) 2
181 (18%) 3 138 (14%) 4 220 (22%)

Scheme: weka.clusterers.SimpleKMeans -N 5 -S 4
Cluster centroids:
Cluster 0 12.507109004739336 14.113744075829384 14.071090047393366 12.739336492890995 13.739336492890995 15.350710900473933

13.85781990521327
Cluster 1 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 2 16.862962962962964 12.540740740740741 13.566666666666666
13.518518518518519 12.981481481481481 13.85925925925926
13.222222222222221
Cluster 3 13.6010101010101 12.262626262626263 13.242424242424242
17.41919191919192 13.080808080808081 13.6010101010101
13.166666666666666
Cluster 4 0.0 0.0 0.0 0.0 0.0 0.0 0.0

0 211 (21%) 1 1 (0%) 2
270 (27%) 3 198 (20%) 4 324 (32%)

Scheme: weka.clusterers.SimpleKMeans -N 5 -S 6
Cluster centroids:
Cluster 0 18.576923076923077 12.451923076923077 13.423076923076923
12.971153846153847 12.73076923076923 13.096153846153847 13.375
Cluster 1 0.0 0.0 0.0 0.0 0.3076923076923077 0.0 0.0
Cluster 2 11.61875 14.175 14.1375 13.1875 13.83125 14.975 14.35
Cluster 3 15.248979591836735 12.979591836734693 13.775510204081632
13.538775510204081 13.151020408163266 14.80408163265306
12.983673469387755
Cluster 4 13.870588235294118 12.052941176470588 13.064705882352941
17.711764705882352 13.147058823529411 13.464705882352941
13.135294117647058

0 104 (10%) 1 325 (32%) 2
160 (16%) 3 245 (24%) 4 170 (17%)

Scheme: weka.clusterers.SimpleKMeans -N 5 -S 8
Cluster centroids:
Cluster 0 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 1 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 2 12.507109004739336 14.113744075829384 14.071090047393366
12.739336492890995 13.739336492890995 15.350710900473933
13.85781990521327
Cluster 3 13.6010101010101 12.262626262626263 13.242424242424242
17.41919191919192 13.080808080808081 13.6010101010101
13.166666666666666
Cluster 4 16.862962962962964 12.540740740740741 13.566666666666666

13.518518518518519 12.981481481481481 13.85925925925926
13.222222222222221

0 1 (0%) 1 324 (32%) 2
211 (21%) 3 198 (20%) 4 270 (27%)

Scheme: weka.clusterers.SimpleKMeans -N 5 -S 10
Cluster centroids:
Cluster 0 11.17094017094017 13.726495726495726 14.068376068376068
15.76923076923077 13.555555555555555 13.35897435897436 14.666666666666666
Cluster 1 13.467336683417086 13.482412060301508 14.020100502512562
12.547738693467336 13.5678391959799 15.844221105527637 13.492462311557789
Cluster 2 0.0 0.0 0.0 0.0 0.3076923076923077 0.0 0.0
Cluster 3 15.151960784313726 12.073529411764707 13.073529411764707
16.754901960784313 12.946078431372548 13.779411764705882
12.691176470588236
Cluster 4 17.654088050314467 12.830188679245284 13.528301886792454
12.748427672955975 13.0 13.50314465408805 13.276729559748428

0 117 (12%) 1 199 (20%) 2
325 (32%) 3 204 (20%) 4 159 (16%)

Scheme: weka.clusterers.SimpleKMeans -N 8 -S -5
Cluster centroids:
Cluster 0 18.666666666666668 12.74074074074074 13.555555555555555
12.049382716049383 12.876543209876543 13.518518518518519
13.160493827160494
Cluster 1 16.817307692307693 11.971153846153847 12.509615384615385
16.41346153846154 12.278846153846153 13.403846153846153
13.182692307692308
Cluster 2 10.027777777777779 15.694444444444445 17.305555555555557
11.055555555555555 13.194444444444445 15.194444444444445 14.0
Cluster 3 11.043478260869565 12.630434782608695 12.826086956521738
18.695652173913043 13.695652173913043 12.41304347826087 15.0
Cluster 4 12.422413793103448 13.724137931034482 12.586206896551724
13.655172413793103 13.801724137931034 15.577586206896552
14.474137931034482
Cluster 5 0.0 0.0 0.0 0.0 0.3076923076923077 0.0 0.0

Cluster 6 14.98136645962733 13.01863354037267 14.372670807453416 12.633540372670808 13.745341614906833 14.453416149068323 13.273291925465838
Cluster 7 14.088888888888889 12.451851851851853 13.837037037037037 16.503703703703703 13.0 14.318518518518518 12.251851851851852

0 81 (8%) 1 104 (10%) 2
36 (4%) 3 46 (5%) 4 116 (12%) 5
325 (32%) 6 161 (16%) 7 135 (13%)

Scheme: weka.clusterers.SimpleKMeans -N 8 -S -4
Cluster centroids:
Cluster 0 16.544159544159545 12.49002849002849 13.452991452991453 14.227920227920228 12.877492877492877 13.897435897435898 13.042735042735043
Cluster 1 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 2 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 3 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 4 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 6 12.432926829268293 13.439024390243903 13.817073170731707 14.612804878048781 13.640243902439025 14.621951219512194 13.789634146341463
Cluster 7 0.0 0.0 0.0 0.0 0.0 0.0 0.0

0 351 (35%) 1 1 (0%) 2
324 (32%) 6 328 (33%)

Scheme: weka.clusterers.SimpleKMeans -N 8 -S -3
Cluster centroids:
Cluster 0 17.984615384615385 12.292307692307693 12.907692307692308 15.446153846153846 11.830769230769231 13.353846153846154 12.907692307692308
Cluster 1 14.719298245614034 11.070175438596491 12.614035087719298 19.403508771929825 13.070175438596491 12.385964912280702 13.070175438596491
Cluster 2 0.0 0.0 0.0 0.0 0.3076923076923077 0.0 0.0
Cluster 3 12.670886075949367 14.063291139240507 13.063291139240507 11.708860759493671 13.227848101265822 17.810126582278482 13.810126582278482

Cluster 4 11.018867924528301 13.943396226415095 14.452830188679245
15.09433962264151 13.764150943396226 13.424528301886792
14.622641509433961
Cluster 5 18.694915254237287 12.898305084745763 13.457627118644067
11.288135593220339 13.067796610169491 13.830508474576272
13.305084745762711
Cluster 6 15.04026845637584 12.758389261744966 14.120805369127517
12.825503355704697 13.979865771812081 13.550335570469798
14.167785234899329
Cluster 7 14.414634146341463 12.871951219512194 13.621951219512194
15.701219512195122 12.939024390243903 14.847560975609756
12.073170731707316

0 65 (6%) 1 57 (6%) 2
325 (32%) 3 79(8%) 4 106 (11%) 5
59 (6%) 6 149 (15%) 7 164 (16%)

Scheme: weka.clusterers.SimpleKMeans -N 8 -S -2
Cluster centroids:
Cluster 0 16.327102803738317 11.467289719626168 12.953271028037383
17.242990654205606 12.476635514018692 13.271028037383177
12.83177570093458
Cluster 1 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 2 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 3 14.615853658536585 13.371951219512194 13.957317073170731
13.768292682926829 12.908536585365853 15.957317073170731
11.908536585365853
Cluster 4 12.446969696969697 12.651515151515152 13.295454545454545
17.136363636363637 13.348484848484848 13.659090909090908
13.780303030303031
Cluster 5 18.565217391304348 12.815217391304348 13.423913043478262
12.358695652173912 12.706521739130435 13.48913043478261
13.217391304347826
Cluster 6 14.70873786407767 12.718446601941748 13.495145631067961
12.495145631067961 14.485436893203884 13.12621359223301
15.29126213592233
Cluster 7 10.802469135802468 14.975308641975309 14.802469135802468
12.320987654320987 13.814814814814815 15.320987654320987
14.382716049382717

0 107 (11%) 1 1 (0%) 2
324 (32%) 3164 (16%) 4 132 (13%) 5
92 (9%) 6 103 (10%) 7 81 (8%)

Scheme: weka.clusterers.SimpleKMeans -N 8 -S -1
Cluster centroids:
Cluster 0 14.676190476190476 13.128571428571428 13.785714285714286
13.533333333333333 13.295238095238096 15.428571428571429
12.642857142857142
Cluster 1 12.470149253731343 12.634328358208956 13.298507462686567
17.119402985074625 13.343283582089553 13.67910447761194
13.776119402985074
Cluster 2 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 3 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 4 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 5 16.31896551724138 11.525862068965518 12.974137931034482
17.060344827586206 12.551724137931034 13.129310344827585
12.956896551724139
Cluster 6 11.330188679245284 14.518867924528301 14.632075471698114
12.330188679245284 14.028301886792454 14.81132075471698
14.69811320754717
Cluster 7 18.035398230088497 12.973451327433628 13.460176991150442
12.079646017699115 13.017699115044248 13.345132743362832
13.619469026548673

0 210 (21%) 1 134 (13%) 2 1 (0%) 3 324 (32%) 5 116 (12%) 6 106
(11%) 7 113 (11%)

Scheme: weka.clusterers.SimpleKMeans -N 8 -S 0
Cluster centroids:
Cluster 0 13.258426966292134 15.617977528089888 15.157303370786517
12.393258426966293 13.44943820224719 15.303370786516854
11.213483146067416
Cluster 1 12.503759398496241 12.74436090225564 13.300751879699249
17.157894736842106 13.353383458646617 13.68421052631579
13.571428571428571
Cluster 2 15.217142857142857 12.09142857142857 13.788571428571428
14.102857142857143 13.36 14.434285714285714 13.548571428571428
Cluster 3 16.476190476190474 11.488095238095237 12.904761904761905
17.738095238095237 12.404761904761905 13.154761904761905

12.357142857142858
Cluster 4 11.460674157303371 13.48314606741573 13.235955056179776
12.696629213483146 13.865168539325843 15.662921348314606
15.910112359550562
Cluster 5 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 6 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 7 18.119266055045873 13.082568807339449 13.403669724770642
12.055045871559633 12.908256880733944 13.458715596330276
13.513761467889909

0 89 (9%) 1 133 (13%) 2
175 (17%) 3 84 (8%) 4 89 (9%) 5
1 (0%) 6 324 (32%) 7 109 (11%)

Scheme: weka.clusterers.SimpleKMeans -N 8 -S 1
Cluster centroids:
Cluster 0 18.0625 12.928571428571429 13.508928571428571
12.098214285714286 13.0 13.348214285714286 13.580357142857142
Cluster 1 16.35135135135135 11.54954954954955 12.864864864864865
17.16216216216216 12.504504504504505 13.17117117117117
12.954954954954955
Cluster 2 12.48062015503876 12.682170542635658 13.310077519379846
17.186046511627907 13.37984496124031 13.55813953488372
13.720930232558139
Cluster 3 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 4 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 5 14.826315789473684 13.031578947368422 14.194736842105263
13.715789473684211 13.25263157894737 15.078947368421053
12.394736842105264
Cluster 6 12.473684210526315 13.342105263157896 11.776315789473685
12.407894736842104 13.723684210526315 16.960526315789473
15.697368421052632
Cluster 7 11.01639344262295 15.344262295081966 16.459016393442624
12.475409836065573 14.147540983606557 13.344262295081966
13.508196721311476

0 112 (11%) 1 111 (11%) 2
129 (13%) 3 1 (0%) 4 324 (32%) 5
190 (19%) 6 76 (8%) 7 61 (6%)

Scheme: weka.clusterers.SimpleKMeans -N 8 -S 2
Cluster centroids:
Cluster 0 14.892307692307693 12.948717948717949 14.343589743589744 13.256410256410257 13.748717948717948 14.189743589743589 13.082051282051282
Cluster 1 12.686567164179104 12.64179104477612 13.111940298507463 17.067164179104477 13.276119402985074 13.940298507462687 13.634328358208956
Cluster 2 16.275 11.566666666666666 12.833333333333334 17.058333333333334 12.4 13.591666666666667 12.841666666666667
Cluster 3 12.592105263157896 13.960526315789474 12.578947368421053 11.802631578947368 13.31578947368421 17.80263157894737 14.328947368421053
Cluster 4 18.5 12.98936170212766 13.319148936170214 12.297872340425531 12.691489361702128 13.48936170212766 13.26595744680851
Cluster 5 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 6 10.533333333333333 15.05 15.866666666666667 13.583333333333334 14.016666666666667 13.116666666666667 14.1
Cluster 7 0.0 0.0 0.0 0.0 0.0 0.0 0.0

0 195 (19%) 1 134 (13%) 2
120 (12%) 3 76 (8%) 4 94 (9%) 5
1 (0%) 6 60 (6%) 7 324 (32%)

Scheme: weka.clusterers.SimpleKMeans -N 8 -S 4
Cluster centroids:
Cluster 0 12.75 14.1125 12.75 11.625 13.3125 17.65 14.1375
Cluster 1 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 2 18.51923076923077 12.759615384615385 13.201923076923077 12.73076923076923 12.740384615384615 13.432692307692308 13.182692307692308
Cluster 3 11.299212598425196 13.88976377952756 14.362204724409448 15.15748031496063 13.614173228346457 13.5748031496063 14.433070866141732
Cluster 4 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 6 14.929936305732484 11.840764331210192 12.92356687898089

17.630573248407643 12.719745222929935 13.668789808917197
12.757961783439491
Cluster 7 14.976303317535544 12.85781990521327 14.255924170616113
13.459715639810426 13.639810426540285 14.19431279620853
13.09478672985782

0 80 (8%) 1 1 (0%) 2
104 (10%) 3127 (13%) 4 324 (32%) 6
157 (16%) 7 211 (21%)

Scheme: weka.clusterers.SimpleKMeans -N 8 -S 5
Cluster centroids:
Cluster 0 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 1 14.649289099526067 13.156398104265403 13.919431279620854
13.23222748815166 13.374407582938389 15.369668246445498
12.763033175355451
Cluster 2 16.27777777777778 11.666666666666666 12.912698412698413
16.936507936507937 12.53968253968254 13.341269841269842
12.865079365079366
Cluster 3 11.227722772277227 14.524752475247524 14.554455445544555
12.386138613861386 14.05940594059406 14.752475247524753
14.841584158415841
Cluster 4 18.221153846153847 12.85576923076923 13.48076923076923
12.192307692307692 12.990384615384615 13.240384615384615
13.557692307692308
Cluster 5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 6 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 7 12.510948905109489 12.715328467153284 13.27007299270073
17.094890510948904 13.291970802919709 13.744525547445255
13.708029197080291

0 1 (0%) 1 211 (21%) 2
126 (13%) 3101 (10%) 4 104 (10%) 5
324 (32%) 7 137 (14%)

Scheme: weka.clusterers.SimpleKMeans -N 8 -S 8
Cluster centroids:
Cluster 0 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 1 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 2 11.159663865546218 14.302521008403362 14.563025210084033

12.991596638655462 13.739495798319327 14.672268907563025
14.873949579831933
Cluster 3 13.974093264248705 12.181347150259068 13.046632124352332
17.471502590673577 13.020725388601036 13.621761658031089
13.1139896373057
Cluster 4 18.02097902097902 12.475524475524475 13.335664335664335
13.258741258741258 12.881118881118882 13.076923076923077
13.552447552447552
Cluster 5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 6 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 7 14.65625 13.191964285714286 13.821428571428571
13.272321428571429 13.410714285714286 15.308035714285714
12.776785714285714

0 1 (0%) 1 324 (32%) 2
119 (12%) 3193 (19%) 4 143 (14%) 7
224 (22%)

Scheme: weka.clusterers.SimpleKMeans -N 8 -S 20
Cluster centroids:
Cluster 0 14.676190476190476 11.60952380952381 12.971428571428572
18.38095238095238 13.076190476190476 13.285714285714286
12.428571428571429
Cluster 1 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 2 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 3 15.341346153846153 12.528846153846153 13.817307692307692
14.533653846153847 13.067307692307692 14.182692307692308
13.048076923076923
Cluster 4 11.388059701492537 13.813432835820896 14.373134328358208
15.097014925373134 13.708955223880597 13.567164179104477
14.365671641791044
Cluster 5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 6 18.28828828828829 12.81981981981982 13.36936936936937
12.396396396396396 12.765765765765765 13.387387387387387
13.495495495495495
Cluster 7 13.198347107438016 13.991735537190083 13.289256198347108
11.859504132231406 13.62809917355372 16.735537190082646
13.710743801652892

0 105 (10%) 1 1 (0%) 2

324 (32%) 3 208 (21%) 4 134 (13%) 6
111 (11%) 7 121 (12%)

Scheme: weka.clusterers.SimpleKMeans -N 12 -S -1
Cluster centroids:
Cluster 0 13.649122807017545 14.929824561403509 14.894736842105264
13.219298245614034 13.043859649122806 15.289473684210526
11.333333333333334
Cluster 1 11.351063829787234 13.053191489361701 13.563829787234043
16.382978723404257 13.627659574468085 13.23404255319149
15.042553191489361
Cluster 2 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 3 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 4 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 5 16.65546218487395 11.92436974789916 13.38655462184874
15.436974789915967 12.436974789915967 13.781512605042018
13.042016806722689
Cluster 6 11.222222222222221 13.796296296296296 13.462962962962964
11.833333333333334 13.296296296296296 17.833333333333332 15.0
Cluster 7 18.857142857142858 12.957142857142857 13.428571428571429
11.857142857142858 12.9 13.428571428571429 13.128571428571428
Cluster 8 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 9 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 10 14.875 12.358333333333333 13.65 12.641666666666667 14.325
13.725 14.858333333333333
Cluster 11 14.527777777777779 12.12037037037037 12.805555555555555
17.75 13.018518518518519 13.86111111111111 12.324074074074074

0 114 (11%) 1 94 (9%) 2 1 (0%) 3 324 (32%) 5 119 (12%) 6 54 (
5%) 7 70 (7%) 10 120 (12%) 11 108 (11%)

Scheme: weka.clusterers.SimpleKMeans -N 12 -S 1
Cluster centroids:
Cluster 0 15.986486486486486 14.054054054054054 13.472972972972974
11.594594594594595 13.68918918918919 14.77027027027027
12.891891891891891
Cluster 1 16.567901234567902 11.283950617283951 13.0 17.59259259259259
12.222222222222221 13.481481481481481 12.382716049382717
Cluster 2 14.367088607594937 10.987341772151899 12.556962025316455
14.39240506329114 13.468354430379748 17.050632911392405

13.620253164556962
Cluster 3 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 4 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 5 13.68695652173913 13.695652173913043 13.565217391304348 16.17391304347826 13.104347826086956 14.008695652173913 12.147826086956522
Cluster 6 10.733333333333333 13.983333333333333 13.433333333333334 12.75 14.2 15.1 16.116666666666667
Cluster 7 13.012820512820513 14.602564102564102 15.551282051282051 12.5 13.0 15.551282051282051 12.179487179487179
Cluster 8 15.329268292682928 12.268292682926829 13.853658536585366 13.71951219512195 14.25609756097561 11.975609756097562 15.073170731707316
Cluster 9 11.266666666666667 12.222222222222221 13.088888888888889 18.644444444444446 13.444444444444445 12.466666666666667 15.133333333333333
Cluster 10 0.0 50.0 50.0 0.0 0.0 0.0 0.0
Cluster 11 19.1875 12.671875 13.40625 12.546875 12.1875 13.578125 13.078125

0 74 (7%) 1 81 (8%) 2 79 (8%) 3 1 (0%) 4 324 (32%) 5 115 (11%) 6 60 (6%) 7 78 (8%) 8 82 (8%) 9 45 (4%) 10 1 (0%) 11 64 (6%)

Scheme: weka.clusterers.SimpleKMeans -N 12 -S 3
Cluster centroids:
Cluster 0 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 1 11.326315789473684 14.589473684210526 14.821052631578947 12.273684210526316 13.91578947368421 14.715789473684211 14.694736842105263
Cluster 2 14.406417112299465 13.085561497326204 13.962566844919786 13.262032085561497 13.342245989304812 15.684491978609625 12.695187165775401
Cluster 3 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 4 11.45 12.7875 13.225 17.1125 13.5125 13.375 14.825
Cluster 5 18.154929577464788 13.112676056338028 13.253521126760564 11.112676056338028 13.056338028169014 13.774647887323944 13.985915492957746
Cluster 6 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 7 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Cluster 8 14.784 12.056 12.744 17.504 12.832 14.064 12.44
Cluster 9 17.041322314049587 12.380165289256198 13.578512396694215 14.834710743801653 12.933884297520661 12.702479338842975 13.198347107438016
Cluster 10 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 11 0.0 0.0 0.0 0.0 0.0 0.0 0.0

0 1 (0%) 1 95 (9%) 2 187 (19%) 3 324 (32%)
4 80 (8%) 5 71 (7%) 8 125 (12%) 9 121 (12%)

Scheme: weka.clusterers.SimpleKMeans -N 20 -S -2
Cluster centroids:
Cluster 0 16.387755102040817 11.938775510204081 14.979591836734693 15.306122448979592 13.244897959183673 11.46938775510204 13.183673469387756
Cluster 1 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 2 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 3 15.85135135135135 11.27027027027027 12.702702702702704 14.594594594594595 12.905405405405405 16.06756756756757 13.135135135135135
Cluster 4 10.64516129032258 13.387096774193548 13.064516129032258 18.838709677419356 13.419354838709678 11.774193548387096 15.096774193548388
Cluster 5 16.44 15.1 12.7 11.26 13.46 13.62 13.74
Cluster 6 16.846153846153847 10.307692307692308 15.307692307692308 11.538461538461538 13.76923076923077 11.615384615384615 17.23076923076923
Cluster 7 0.0 29.0 29.0 0.0 0.0 20.5 20.5
Cluster 8 20.642857142857142 12.285714285714286 13.357142857142858 9.928571428571429 12.5 14.5 13.285714285714286
Cluster 9 11.425 10.95 13.35 15.55 14.1 16.525 14.475
Cluster 10 16.09090909090909 10.878787878787879 11.93939393939394 19.515151515151516 12.515151515151516 12.909090909090908 12.696969696969697
Cluster 11 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 12 13.365853658536585 13.926829268292684 12.317073170731707 11.878048780487806 13.414634146341463 18.75609756097561 12.634146341463415
Cluster 13 11.636363636363637 13.787878787878787 15.242424242424242 11.575757575757576 13.636363636363637 13.969696969696969

16.393939393939394
Cluster 14 19.022727272727273 12.386363636363637 12.886363636363637 14.181818181818182 12.25 13.454545454545455 12.454545454545455
Cluster 15 14.714285714285714 13.476190476190476 16.126984126984127 13.428571428571429 12.158730158730158 14.88888888888889 11.777777777777779
Cluster 16 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 17 14.025316455696203 13.316455696202532 13.30379746835443 16.72151898734177 13.544303797468354 14.025316455696203 11.49367088607595
Cluster 18 11.88679245283019 15.452830188679245 14.39622641509434 13.528301886792454 14.49056603773585 13.962264150943396 12.584905660377359
Cluster 19 13.783333333333333 12.516666666666667 12.716666666666667 14.633333333333333 13.783333333333333 13.116666666666667 15.916666666666666

0 49 (5%) 1 1 (0%) 2 324 (32%) 3 74 (7%) 4 31 (3%) 5 50 (5%) 6 13 (1%) 7 2 (0%) 8 14 (1%) 9 40 (4%) 10 33 (3%) 12 41 (4%) 13
33 (3%)14 44 (4%) 15 63 (6%) 17
79 (8%) 18 53 (5%) 19 60 (6%)

Scheme: weka.clusterers.SimpleKMeans -N 20 -S 0
Cluster centroids:
Cluster 0 13.85 15.133333333333333 13.833333333333334 14.633333333333333 13.633333333333333 14.3 10.983333333333333
Cluster 1 13.814814814814815 11.796296296296296 13.944444444444445 17.055555555555557 13.444444444444445 14.537037037037036 11.777777777777779
Cluster 2 14.719298245614034 11.070175438596491 15.666666666666666 13.842105263157896 12.526315789473685 15.719298245614034 13.052631578947368
Cluster 3 16.76923076923077 9.923076923076923 13.961538461538462 18.884615384615383 12.615384615384615 10.923076923076923 13.5
Cluster 4 13.895833333333334 12.979166666666666 12.229166666666666 15.958333333333334 12.8125 13.208333333333334 15.270833333333334

Cluster 5 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 6 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 7 18.677419354838708 13.67741935483871 12.580645161290322
10.35483870967742 13.193548387096774 13.516129032258064
14.612903225806452
Cluster 8 16.333333333333332 12.933333333333334 10.866666666666667
18.266666666666666 12.066666666666666 15.733333333333333
10.466666666666667
Cluster 9 16.52 12.02 12.08 15.48 12.0 14.98 13.46
Cluster 10 19.37837837837838 12.216216216216216 13.675675675675675
13.783783783783784 12.216216216216216 13.513513513513514
11.91891891891892
Cluster 11 11.057692307692308 13.76923076923077 13.576923076923077
13.442307692307692 14.153846153846153 15.153846153846153
15.25
Cluster 12 16.083333333333332 13.916666666666666 13.708333333333334
11.791666666666666 13.3125 15.291666666666666
12.291666666666666
Cluster 13 0.0 8.0 8.0 0.0 0.0 41.0 41.0
Cluster 14 11.727272727272727 11.090909090909092 12.0 22.454545454545453
12.909090909090908 11.545454545454545 14.454545454545455
Cluster 15 13.976744186046512 12.279069767441861 13.55813953488372
12.372093023255815 15.069767441860465 12.883720930232558
16.302325581395348
Cluster 16 10.829268292682928 13.170731707317072 13.585365853658537
17.121951219512194 13.536585365853659 13.170731707317072
14.78048780487805
Cluster 17 13.085106382978724 14.148936170212766 12.574468085106384
11.851063829787234 13.25531914893617 18.46808510638298
12.893617021276595
Cluster 18 16.22222222222222 12.822222222222223 14.466666666666667
13.844444444444445 14.177777777777777 11.355555555555556
13.555555555555555
Cluster 19 10.23076923076923 18.46153846153846 21.46153846153846
10.538461538461538 13.076923076923077 11.538461538461538
11.384615384615385

0 60 (6%) 1 54 (5%) 2 57 (6%) 3 26 (3%) 4 48 (5%)
5 1 (0%) 6 324 (32%) 7 31 (3%) 8 15 (1%) 9 50 (5%) 10 37 (4%)
11 52 (5%) 12

48 (5%)13 1 (0%) 14 11 (1%) 15
43 (4%) 16 41 (4%) 17
47 (5%) 18 45 (4%) 19 13 (1%)

Scheme: weka.clusterers.SimpleKMeans -N 20 -S 3
Cluster centroids:
Cluster 0 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 1 11.6875 13.9375 14.03125 12.25 13.90625 14.21875 16.21875
Cluster 2 9.909090909090908 20.454545454545453 21.454545454545453 10.454545454545455 11.909090909090908 12.636363636363637 10.090909090909092
Cluster 3 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 4 11.220588235294118 12.926470588235293 13.279411764705882 16.455882352941178 13.691176470588236 13.779411764705882 14.941176470588236
Cluster 5 16.916666666666668 13.0625 12.958333333333334 10.770833333333334 13.375 13.666666666666666 15.729166666666666
Cluster 6 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 7 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 8 14.5 10.935897435897436 14.346153846153847 14.166666666666666 12.782051282051283 16.0 13.820512820512821
Cluster 9 15.78888888888889 13.311111111111112 13.88888888888889 13.566666666666666 14.266666666666667 12.744444444444444 12.877777777777778
Cluster 10 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 11 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 12 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 13 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 14 14.182692307692308 13.182692307692308 13.740384615384615 16.557692307692307 13.14423076923077 14.01923076923077 11.615384615384615
Cluster 15 13.443037974683545 14.544303797468354 13.227848101265822 12.620253164556962 13.39240506329114 17.20253164556962 11.936708860759493
Cluster 16 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 17 16.802816901408452 11.169014084507042 12.070422535211268 17.070422535211268 12.28169014084507 13.394366197183098 13.76056338028169

Cluster 18 12.875 10.625 12.625 22.0625 12.625 11.4375 14.0
Cluster 19 19.3 12.7 13.86 12.84 12.4 13.72 11.84

0 1 (0%) 1 64 (6%) 2 11 (1%) 3 324 (32%)
4 68 (7%) 5 48 (5%) 8 78 (8%) 9 90 (9%) 14 104 (10%) 15 79 (8%) 17
71 (7%) 18 16 (2%) 19 50 (5%)

Scheme: weka.clusterers.SimpleKMeans -N 20 -S 10
Cluster centroids:
Cluster 0 11.216216216216216 10.64864864864865 12.837837837837839 16.37837837837838 14.135135135135135 16.054054054054053 15.08108108108108
Cluster 1 13.409090909090908 14.113636363636363 11.727272727272727 13.068181818181818 13.386363636363637 14.090909090909092 16.522727272727273
Cluster 2 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 3 16.344827586206897 11.275862068965518 11.413793103448276 17.06896551724138 12.137931034482758 15.241379310344827 12.982758620689655
Cluster 4 16.8125 11.583333333333334 14.083333333333334 15.479166666666666 12.895833333333334 11.625 14.1875
Cluster 5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 6 0.0 29.0 29.0 0.0 0.0 20.5 20.5
Cluster 7 19.53846153846154 13.461538461538462 14.923076923076923 9.0 13.692307692307692 12.846153846153847 13.23076923076923
Cluster 8 18.80952380952381 12.714285714285714 11.857142857142858 12.476190476190476 12.0 12.666666666666666 16.047619047619047
Cluster 9 14.64864864864865 12.378378378378379 14.72972972972973 12.108108108108109 15.837837837837839 11.35135135135135 15.216216216216216
Cluster 10 12.782051282051283 13.474358974358974 13.461538461538462 16.602564102564102 13.243589743589743 13.35897435897436 13.564102564102564
Cluster 11 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 12 10.435897435897436 15.23076923076923 15.487179487179487 12.948717948717949 14.41025641025641 13.487179487179487 14.256410256410257

Cluster 13 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 14 13.058823529411764 10.764705882352942 12.529411764705882 21.941176470588236 12.647058823529411 11.294117647058824 14.058823529411764
Cluster 15 14.875 13.765625 14.09375 16.140625 13.359375 13.5625 10.5625
Cluster 16 12.891304347826088 14.152173913043478 12.673913043478262 12.021739130434783 13.347826086956522 18.52173913043478 12.717391304347826
Cluster 17 18.743589743589745 13.102564102564102 13.564102564102564 13.205128205128204 12.35897435897436 14.23076923076923 11.41025641025641
Cluster 18 14.058823529411764 12.794117647058824 15.794117647058824 13.779411764705882 12.573529411764707 15.264705882352942 12.220588235294118
Cluster 19 15.823529411764707 12.764705882352942 13.588235294117647 12.279411764705882 13.632352941176471 15.455882352941176 12.926470588235293

0 37 (4%) 1 44 (4%) 2 1 (0%) 3 58 (6%) 4 48 (5%) 5 324 (32%) 6 2 (0%) 7 13 (1%) 8 21 (2%) 9 37 (4%) 10 78 (8%) 12 39 (4%) 14
17 (2%) 15 64 (6%) 16 46 (5%) 17
39 (4%) 18 68 (7%) 19 68 (7%)

Scheme: weka.clusterers.SimpleKMeans -N 100 -S -1
Cluster centroids:
Cluster 0 12.75 13.833333333333334 14.291666666666666 13.916666666666666 12.875 16.166666666666668
12.375
Cluster 1 13.2 15.066666666666666 12.066666666666666 15.333333333333334 12.8 13.466666666666667
14.333333333333334
Cluster 2 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 3 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 4 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 5 16.928571428571427 11.357142857142858 14.071428571428571 13.5 13.142857142857142 13.857142857142858 13.571428571428571
Cluster 6 12.666666666666666 12.555555555555555 9.222222222222221 12.555555555555555 14.222222222222221 21.22222222222222

13.666666666666666
Cluster 7 19.666666666666668 8.333333333333334 14.0 15.333333333333334 11.0 17.666666666666668 11.333333333333334
Cluster 8 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 9 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 10 17.88888888888889 15.055555555555555 12.88888888888889 13.055555555555555 13.166666666666666 12.444444444444445 11.833333333333334
Cluster 11 16.866666666666667 12.333333333333334 11.133333333333333 15.2 12.2 13.933333333333334 14.866666666666667
Cluster 12 16.666666666666668 12.0 14.0 8.333333333333334 18.0 13.666666666666666 14.0
Cluster 13 10.8125 15.0625 16.125 12.875 13.6875 14.75 13.1875
Cluster 14 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 15 14.333333333333334 10.666666666666666 11.583333333333334 13.416666666666666 16.5 15.666666666666666 14.416666666666666
Cluster 16 8.5 10.5 16.5 16.0 14.0 18.0 13.0
Cluster 17 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 18 14.857142857142858 10.571428571428571 14.571428571428571 11.142857142857142 13.857142857142858 17.142857142857142 14.571428571428571
Cluster 19 18.833333333333332 11.0 12.0 16.0 11.833333333333334 15.166666666666666 12.5
Cluster 20 14.25 13.0 15.0 16.75 11.5 18.0 8.5
Cluster 21 15.444444444444445 11.11111111111111 13.777777777777779 14.11111111111111 16.666666666666668 11.444444444444445 13.777777777777779
Cluster 22 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 23 14.538461538461538 12.461538461538462 19.0 12.923076923076923 10.538461538461538 14.538461538461538 12.76923076923077
Cluster 24 20.88888888888889 12.666666666666666 15.11111111111111 9.333333333333334 12.444444444444445 13.666666666666666 12.555555555555555
Cluster 25 8.25 16.25 11.5 17.75 13.25 10.75 18.25
Cluster 26 14.8 18.0 15.8 13.8 9.4 15.6 9.4
Cluster 27 11.9 9.1 12.0 16.9 14.6 17.6 14.4
Cluster 28 13.818181818181818 12.181818181818182 13.363636363636363 17.363636363636363 14.636363636363637 12.545454545454545 12.545454545454545

Cluster 29 12.285714285714286 9.571428571428571 14.428571428571429 16.857142857142858 12.0 14.428571428571429 16.428571428571427
Cluster 30 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 31 19.75 13.25 11.125 12.5 12.875 16.0 11.0
Cluster 32 12.75 13.375 17.25 10.625 12.5 15.0 14.875
Cluster 33 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 34 15.0 15.0 12.583333333333334 17.25 13.75 12.666666666666666 10.25
Cluster 35 16.75 10.375 14.125 12.5 12.625 17.75 12.375
Cluster 36 10.8 14.7 17.1 12.7 13.1 10.8 16.9
Cluster 37 11.666666666666666 17.333333333333332 16.166666666666668 9.833333333333334 12.666666666666666 17.833333333333332 11.166666666666666
Cluster 38 12.0 11.8 12.0 14.4 13.6 17.2 15.8
Cluster 39 9.0 12.166666666666666 13.166666666666666 14.666666666666666 16.833333333333332 16.0 14.833333333333334
Cluster 40 0.0 8.0 8.0 0.0 0.0 41.0 41.0
Cluster 41 13.7 11.1 14.85 17.2 13.45 15.05 11.35
Cluster 42 15.0 7.5 14.0 16.0 11.0 17.5 16.0
Cluster 43 13.875 17.375 11.125 11.5 11.875 16.75 13.125
Cluster 44 16.5 9.666666666666666 16.166666666666668 11.166666666666666 14.0 10.833333333333334 18.333333333333332
Cluster 45 15.666666666666666 14.533333333333333 15.333333333333334 10.866666666666667 12.733333333333333 14.733333333333333 12.2
Cluster 46 20.333333333333332 12.666666666666666 11.5 14.833333333333334 12.0 10.166666666666666 15.166666666666666
Cluster 47 15.8 13.933333333333334 14.333333333333334 13.133333333333333 14.8 10.4 13.933333333333334
Cluster 48 12.5 13.5 14.666666666666666 16.666666666666668 12.083333333333334 11.666666666666666 15.416666666666666
Cluster 49 15.5 12.5 11.166666666666666 16.333333333333332 10.333333333333334 16.666666666666668 14.0
Cluster 50 15.0 18.0 9.0 14.0 18.0 13.0 9.0
Cluster 51 19.6 10.6 11.2 11.6 11.8 14.4 17.0
Cluster 52 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 53 15.105263157894736 11.157894736842104 14.263157894736842 14.368421052631579 12.31578947368421 16.473684210526315 12.894736842105264

Cluster 54 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 55 11.166666666666666 12.916666666666666 15.0 15.75 15.083333333333334 13.0 13.0
Cluster 56 13.545454545454545 11.954545454545455 14.727272727272727 14.227272727272727 13.5 13.909090909090908 14.409090909090908
Cluster 57 18.0 20.0 12.5 9.5 13.5 11.5 12.0
Cluster 58 13.5 12.125 13.75 11.5 17.5 12.5 15.5
Cluster 59 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 60 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 61 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 62 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 63 15.75 13.392857142857142 14.678571428571429 14.607142857142858 12.857142857142858 13.607142857142858 11.678571428571429
Cluster 64 14.25 13.125 10.125 17.75 12.625 13.875 14.625
Cluster 65 16.0 9.75 14.75 20.125 11.625 12.5 11.5
Cluster 66 12.692307692307692 14.76923076923077 14.384615384615385 12.846153846153847 17.076923076923077 12.307692307692308 12.153846153846153
Cluster 67 16.6875 15.0 11.3125 10.8125 12.625 14.25 16.0625
Cluster 68 14.7 13.1 11.5 13.7 12.8 12.8 18.1
Cluster 69 18.666666666666668 8.666666666666666 14.0 18.166666666666668 12.5 10.666666666666666 14.333333333333334
Cluster 70 0.0 50.0 50.0 0.0 0.0 0.0 0.0
Cluster 71 16.6 10.2 9.6 17.8 11.4 18.8 12.0
Cluster 72 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 73 11.555555555555555 14.666666666666666 11.333333333333334 11.222222222222221 13.88888888888889 16.333333333333332 17.22222222222222
Cluster 74 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 75 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 76 13.153846153846153 15.76923076923077 12.923076923076923 15.538461538461538 14.307692307692308 14.307692307692308 10.153846153846153
Cluster 77 15.333333333333334 15.666666666666666 12.444444444444445 12.777777777777779 13.88888888888889 15.666666666666666 10.666666666666666
Cluster 78 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 79 19.0 11.25 16.625 13.375 12.5 12.0 12.125

Cluster 80 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 81 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 82 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 83 16.7 12.6 10.2 19.0 13.2 12.8 12.2
Cluster 84 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 85 14.181818181818182 13.090909090909092 14.454545454545455 12.363636363636363 13.090909090909092 18.727272727272727 10.636363636363637
Cluster 86 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 87 12.3 10.5 11.7 22.6 13.0 11.5 14.5
Cluster 88 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 89 10.454545454545455 14.727272727272727 13.363636363636363 18.181818181818183 13.272727272727273 12.636363636363637 13.818181818181818
Cluster 90 12.142857142857142 14.285714285714286 10.0 15.142857142857142 14.428571428571429 12.428571428571429 17.857142857142858
Cluster 91 16.0 12.25 12.25 15.0 13.5625 15.8125 11.375
Cluster 92 14.789473684210526 13.105263157894736 13.263157894736842 16.31578947368421 11.789473684210526 13.842105263157896 13.0
Cluster 93 16.375 10.5 18.625 16.625 13.0 10.625 11.0
Cluster 94 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 95 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 96 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 97 15.666666666666666 10.444444444444445 12.444444444444445 16.333333333333332 13.555555555555555 11.555555555555555 16.444444444444443
Cluster 98 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 99 0.0 0.0 0.0 0.0 0.0 0.0 0.0

0 24 (2%) 1 15 (1%) 2 1 (0%) 3 324 (32%) 5 14 (1%) 6 9 (1%) 7 3 (0%) 10 18 (2%) 11 15 (1%) 12 3 (0%) 13 16 (2%) 15 12 (1%) 16 2 (0%) 18 7 (1%) 19 6 (1%) 20 4 (0%) 21 9 (1%) 23 13 (1%) 24 9 (1%) 25 4 (0%) 26 5 (0%) 27 10 (1%) 28 11 (1%) 29 7 (1%) 31 8 (1%) 32 8 (1%) 34 12 (1%) 35 8 (1%) 36 10 (1%) 37 6 (1%) 38 10 (1%) 39 6 (1%) 40 1 (0%) 41 20 (2%) 42 2 (0%) 43 8 (1%) 44 6 (1%) 45 15 (1%) 46 6 (1%) 47 15 (1%) 48 12 (1%) 49 6 (1%) 50 1 (0%) 51 5 (0%) 53 19 (2%) 55 12 (1%) 56 22 (2%) 57 2 (0%) 58 8 (1%) 63 28 (3%) 64 8 (1%) 65 8 (1%) 66 13 (1%) 67 16 (2%) 68 10 (1%) 69 6 (1%) 70 1 (0%) 71 5 (0%) 73 9 (1%) 76 13 (1%) 77 9 (1%) 79 8 (1%) 83 10 (1%) 85 11 (1%) 87 10 (

1%) 89 11 (1%) 90 7 (1%) 91 16 (2%) 92 19 (2%) 93 8 (1%) 97 9 (1%)

Scheme: weka.clusterers.SimpleKMeans -N 100 -S 0
Cluster centroids:
Cluster 0 13.142857142857142 16.714285714285715 11.714285714285714 15.571428571428571 15.142857142857142 14.142857142857142 9.714285714285714
Cluster 1 13.857142857142858 11.619047619047619 15.0 16.952380952380953 12.714285714285714 13.857142857142858 12.380952380952381
Cluster 2 15.083333333333334 11.583333333333334 14.583333333333334 15.0 13.166666666666666 15.416666666666666 11.25
Cluster 3 17.0 12.5 9.0 20.0 11.5 11.5 15.5
Cluster 4 13.894736842105264 13.842105263157896 13.263157894736842 15.578947368421053 12.31578947368421 13.789473684210526 13.631578947368421
Cluster 5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 6 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 7 18.0 18.666666666666668 12.666666666666666 10.333333333333334 14.0 11.666666666666666 12.0
Cluster 8 15.0 14.0 14.0 17.0 12.0 18.0 7.0
Cluster 9 16.363636363636363 13.181818181818182 10.909090909090908 14.636363636363637 11.454545454545455 13.454545454545455 16.454545454545453
Cluster 10 18.2 17.4 10.6 15.0 13.0 11.2 10.6
Cluster 11 12.363636363636363 11.818181818181818 12.909090909090908 15.0 12.545454545454545 15.272727272727273 16.09090909090909
Cluster 12 15.608695652173912 13.347826086956522 13.304347826086957 13.565217391304348 14.043478260869565 15.08695652173913 11.73913043478261
Cluster 13 0.0 8.0 8.0 0.0 0.0 41.0 41.0
Cluster 14 15.8 10.2 16.6 20.4 11.8 10.8 10.0
Cluster 15 14.3 11.1 11.5 13.5 15.9 15.2 15.0
Cluster 16 13.307692307692308 11.76923076923077 15.692307692307692 14.692307692307692 12.23076923076923 15.615384615384615 12.923076923076923
Cluster 17 14.3 13.1 15.3 11.8 13.1 18.5 10.6
Cluster 18 16.166666666666668 10.166666666666666 18.333333333333332 16.5 12.5 11.0 12.5
Cluster 19 11.4 14.6 17.2 11.2 12.2 10.4 19.2

Cluster 20 16.846153846153847 10.307692307692308 12.0 18.846153846153847 13.307692307692308 13.461538461538462 11.846153846153847
Cluster 21 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 22 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 23 9.0 16.0 13.444444444444445 17.22222222222222 13.11111111111111 12.0 15.555555555555555
Cluster 24 13.444444444444445 14.666666666666666 10.11111111111111 17.333333333333332 13.0 13.777777777777779 14.11111111111111
Cluster 25 19.11111111111111 10.222222222222221 12.777777777777779 17.22222222222222 12.777777777777779 9.777777777777779 15.222222222222221
Cluster 26 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 27 13.0 13.133333333333333 16.066666666666666 12.133333333333333 12.466666666666667 16.2 13.533333333333333
Cluster 28 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 29 14.7 12.1 13.9 13.5 17.5 10.5 13.9
Cluster 30 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 31 12.88888888888889 13.666666666666666 11.777777777777779 11.444444444444445 14.11111111111111 14.333333333333334 18.333333333333332
Cluster 32 13.428571428571429 14.428571428571429 10.714285714285714 13.0 13.428571428571429 19.571428571428573 11.428571428571429
Cluster 33 16.210526315789473 12.0 11.578947368421053 15.842105263157896 11.631578947368421 16.05263157894737 13.105263157894736
Cluster 34 13.444444444444445 14.777777777777779 15.11111111111111 13.11111111111111 16.88888888888889 13.11111111111111 10.11111111111111
Cluster 35 15.785714285714286 15.071428571428571 14.571428571428571 10.714285714285714 12.428571428571429 14.642857142857142 13.071428571428571
Cluster 36 12.3 12.3 10.3 13.4 15.4 18.8 14.1
Cluster 37 20.875 12.5 14.75 10.75 12.5 13.75 11.625
Cluster 38 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 39 12.466666666666667 13.466666666666667 14.866666666666667 16.333333333333332 13.133333333333333 11.066666666666666 15.066666666666666
Cluster 40 12.833333333333334 9.166666666666666 12.333333333333334 16.166666666666668 14.333333333333334 17.166666666666668

14.333333333333334
Cluster 41 11.0 13.333333333333334 14.583333333333334 16.416666666666668
14.166666666666666 14.333333333333334 12.25
Cluster 42 10.0 7.0 16.0 17.0 14.0 14.0 18.0
Cluster 43 13.545454545454545 15.272727272727273 13.181818181818182
14.909090909090908 12.909090909090908 14.818181818181818
11.727272727272727
Cluster 44 16.846153846153847 10.923076923076923 13.846153846153847
14.23076923076923 13.76923076923077 12.692307692307692
14.153846153846153
Cluster 45 15.8 10.0 13.2 12.4 14.0 16.0 15.4
Cluster 46 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 47 14.5 18.25 13.125 11.375 12.75 16.25 10.25
Cluster 48 15.4 9.4 13.6 17.8 13.4 16.2 11.2
Cluster 49 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 50 12.333333333333334 15.5 20.666666666666668
11.333333333333334 12.166666666666666 12.5 12.0
Cluster 51 16.181818181818183 14.181818181818182 15.090909090909092
12.636363636363637 13.272727272727273 10.909090909090908 14.0
Cluster 52 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 53 19.4 11.8 15.5 13.3 12.8 11.3 12.7
Cluster 54 15.75 9.5 15.75 11.25 14.25 11.25 19.0
Cluster 55 16.238095238095237 13.761904761904763 14.142857142857142
15.19047619047619 12.0 13.142857142857142 12.142857142857142
Cluster 56 13.333333333333334 11.333333333333334 8.333333333333334
14.666666666666666 11.666666666666666 23.666666666666668
13.333333333333334
Cluster 57 17.833333333333332 16.166666666666668 11.0 12.666666666666666
12.333333333333334 14.666666666666666 11.833333333333334
Cluster 58 10.333333333333334 16.5 13.5 11.0 13.5 18.333333333333332
13.0
Cluster 59 18.11111111111111 12.333333333333334 15.88888888888889
14.11111111111111 11.555555555555555 14.88888888888889 10.0
Cluster 60 19.666666666666668 11.0 11.166666666666666 12.0
11.333333333333334 14.0 17.0
Cluster 61 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 62 11.6 12.8 10.8 15.0 15.6 12.6 17.4
Cluster 63 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 64 18.666666666666668 10.0 17.333333333333332
10.333333333333334 12.0 11.0 17.0

Cluster 65 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 66 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 67 15.333333333333334 7.0 14.0 13.666666666666666
16.333333333333332 17.0 12.666666666666666
Cluster 68 11.761904761904763 15.333333333333334
14.19047619047619 13.523809523809524 14.142857142857142
14.142857142857142 13.285714285714286
Cluster 69 12.714285714285714 11.142857142857142 13.714285714285714
17.714285714285715 13.285714285714286 16.714285714285715
11.285714285714286
Cluster 70 16.6 12.8 11.4 14.2 18.0 10.4 13.4
Cluster 71 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 72 19.333333333333332 10.555555555555555 11.11111111111111
15.444444444444445 11.666666666666666 17.0 11.777777777777779
Cluster 73 15.333333333333334 11.066666666666666 14.333333333333334
14.133333333333333 11.133333333333333 16.6 14.133333333333333
Cluster 74 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 75 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 76 14.125 13.0 12.75 10.75 13.25 17.75 14.375
Cluster 77 14.5 11.5 14.0 17.5 12.0 18.0 9.5
Cluster 78 14.714285714285714 14.142857142857142 18.571428571428573
14.285714285714286 8.857142857142858 15.714285714285714 10.0
Cluster 79 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 80 17.416666666666668 14.5 11.583333333333334 10.25
12.916666666666666 13.75 16.166666666666668
Cluster 81 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 82 13.125 17.875 9.875 11.75 12.25 15.0 16.375
Cluster 83 17.363636363636363 11.909090909090908 14.0
12.181818181818182 13.545454545454545 14.454545454545455
12.636363636363637
Cluster 84 11.571428571428571 11.571428571428571 11.857142857142858
23.0 12.714285714285714 11.571428571428571 13.714285714285714
Cluster 85 13.533333333333333 13.466666666666667 13.266666666666667
13.066666666666666 14.533333333333333 13.266666666666667
15.066666666666666
Cluster 86 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 87 21.5 15.5 11.0 8.5 13.0 17.5 9.5
Cluster 88 14.68421052631579 14.052631578947368 12.052631578947368
17.63157894736842 14.052631578947368 12.894736842105264
11.052631578947368

Cluster 89 11.125 11.125 12.5 19.5 13.5 12.75 16.25
Cluster 90 14.714285714285714 10.571428571428571 12.357142857142858 16.642857142857142 13.571428571428571 12.857142857142858 15.785714285714286
Cluster 91 13.428571428571429 10.285714285714286 16.714285714285715 12.857142857142858 15.714285714285714 12.571428571428571 14.714285714285714
Cluster 92 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 93 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 94 15.0 7.5 10.0 22.5 13.0 11.5 17.0
Cluster 95 9.444444444444445 10.555555555555555 12.666666666666666 14.88888888888889 16.444444444444443 17.555555555555557 15.0
Cluster 96 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 97 0.0 50.0 50.0 0.0 0.0 0.0 0.0
Cluster 98 9.777777777777779 14.777777777777779 17.88888888888889 12.777777777777779 13.555555555555555 13.333333333333334 14.11111111111111
Cluster 99 15.666666666666666 10.5 17.0 11.0 15.166666666666666 14.333333333333334 13.0

0 7 (1%) 1 21 (2%) 2 12 (1%) 3 2 (0%) 4 19 (2%) 5 324 (32%) 7 3 (0%) 8 1 (0%) 9 11 (1%) 10 5 (0%) 11 11 (1%) 12 23 (2%) 13 1 (0%) 14 5 (0%) 15 10 (1%) 16 13 (1%) 17 10 (1%) 18 6 (1%) 19 5 (0%) 20 13 (1%) 23 9 (1%) 24 9 (1%) 25 9 (1%) 27 15 (1%) 29 10 (1%) 31 9 (1%) 32 7 (1%) 33 19 (2%) 34 9 (1%) 35 14 (1%) 36 10 (1%) 37 8 (1%) 39 15 (1%) 40 6 (1%) 41 12 (1%) 42 1 (0%) 43 11 (1%) 44 13 (1%) 45 5 (0%) 47 8 (1%) 48 5 (0%) 50 6 (1%) 51 11 (1%) 53 10 (1%) 54 4 (0%) 55 21 (2%) 56 3 (0%) 57 6 (1%) 58 6 (1%) 59 9 (1%) 60 6 (1%) 62 5 (0%) 64 3 (0%) 67 3 (0%) 68 21 (2%) 69 7 (1%) 70 5 (0%) 72 9 (1%) 73 15 (1%) 76 8 (1%) 77 2 (0%) 78 7 (1%) 80 12 (1%) 82 8 (1%) 83 11 (1%) 84 7 (1%) 85 15 (1%) 87 2 (0%) 88 19 (2%) 89 8 (1%) 90 14 (1%) 91 7 (1%) 94 2 (0%) 95 9 (1%) 96 1 (0%) 97 1 (0%) 98 9 (1%) 99 6 (1%)

Scheme: weka.clusterers.SimpleKMeans -N 100 -S 10
Cluster centroids:
Cluster 0 12.285714285714286 9.928571428571429 12.642857142857142 15.714285714285714 14.0 17.428571428571427 14.571428571428571
Cluster 1 13.444444444444445 14.444444444444445 10.11111111111111

11.88888888888889 13.88888888888889 14.0 18.88888888888889
Cluster 2 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 3 16.38888888888889 11.222222222222221 11.0 15.88888888888889
12.88888888888889 15.944444444444445 13.222222222222221
Cluster 4 18.818181818181817 10.272727272727273 13.181818181818182
15.090909090909092 11.0 16.454545454545453 12.181818181818182
Cluster 5 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 6 10.3 14.7 18.2 12.0 14.3 11.7 14.9
Cluster 7 20.666666666666668 13.0 13.0 7.333333333333333 15.0 15.0
12.333333333333334
Cluster 8 19.2 11.6 10.2 12.2 13.2 15.0 14.6
Cluster 9 13.333333333333334 12.066666666666666 15.0 12.0
16.066666666666666 11.866666666666667 15.933333333333334
Cluster 10 14.0 13.136363636363637 13.136363636363637
16.136363636363637 12.954545454545455 14.045454545454545
13.0
Cluster 11 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 12 11.666666666666666 16.77777777777778 16.77777777777778
10.444444444444445 13.777777777777779 16.11111111111111
10.88888888888889
Cluster 13 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 14 15.454545454545455 11.454545454545455 13.545454545454545
17.818181818181817 12.090909090909092 12.181818181818182
13.454545454545455
Cluster 15 14.428571428571429 16.285714285714285 14.142857142857142
14.428571428571429 14.857142857142858 13.0 9.0
Cluster 16 13.11111111111111 14.777777777777779 13.555555555555555
12.61111111111111 13.5 15.555555555555555 13.11111111111111
Cluster 17 20.25 13.75 12.25 10.75 13.25 14.25 11.5
Cluster 18 14.7 13.1 19.5 13.0 10.0 14.6 11.7
Cluster 19 14.428571428571429 13.285714285714286 14.214285714285714
13.714285714285714 14.785714285714286 12.642857142857142
13.571428571428571
Cluster 20 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 21 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 22 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 23 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 24 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 25 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 26 21.0 11.666666666666666 13.666666666666666 12.666666666666666

11.666666666666666 16.333333333333332 10.0
Cluster 27 11.142857142857142 9.428571428571429 12.428571428571429
18.857142857142858 13.714285714285714 14.285714285714286
16.857142857142858
Cluster 28 18.714285714285715 11.0 17.285714285714285
13.428571428571429 12.571428571428571 11.714285714285714
12.428571428571429
Cluster 29 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 30 13.0 12.166666666666666 13.833333333333334 20.666666666666668
13.166666666666666 11.5 11.833333333333334
Cluster 31 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 32 13.333333333333334 11.333333333333334 8.333333333333334
14.666666666666666 11.666666666666666 23.666666666666668
13.333333333333334
Cluster 33 14.23076923076923 12.76923076923077 14.692307692307692
12.0
13.0 18.692307692307693 11.307692307692308
Cluster 34 14.6 10.2 12.5 16.3 13.5 13.3 16.3
Cluster 35 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 36 16.363636363636363 13.636363636363637 11.636363636363637
14.545454545454545 10.909090909090908 12.909090909090908
16.545454545454547
Cluster 37 13.214285714285714 14.357142857142858 14.071428571428571
14.357142857142858 14.0 15.928571428571429 10.357142857142858
Cluster 38 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 39 16.75 11.4375 12.9375 14.6875 13.8125 12.5 14.0625
Cluster 40 16.38095238095238 14.619047619047619 12.428571428571429
10.523809523809524 13.095238095238095 14.285714285714286
15.142857142857142
Cluster 41 9.11111111111111 12.88888888888889 12.333333333333334
13.666666666666666 16.666666666666668 17.444444444444443
14.444444444444445
Cluster 42 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 43 12.733333333333333 12.8 14.533333333333333 15.666666666666666
12.2 12.866666666666667 15.4
Cluster 44 15.214285714285714 11.928571428571429 15.642857142857142
15.285714285714286 13.0 13.785714285714286 11.571428571428571
Cluster 45 15.5 9.5 12.333333333333334 21.5 12.666666666666666
11.0 14.166666666666666
Cluster 46 14.25 10.5625 12.375 13.3125 15.25 15.5625 15.0625

Cluster 47 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 48 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 49 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 50 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 51 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 52 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 53 13.11111111111111 16.333333333333332 10.777777777777779 15.0 12.333333333333334 14.777777777777779 13.88888888888889
Cluster 54 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 55 12.0 18.0 12.857142857142858 12.857142857142858 12.285714285714286 12.857142857142858 15.571428571428571
Cluster 56 12.727272727272727 13.090909090909092 9.363636363636363 12.090909090909092 15.272727272727273 19.454545454545453 14.090909090909092
Cluster 57 10.4 11.4 10.8 23.8 12.2 11.8 15.8
Cluster 58 15.333333333333334 14.75 13.25 17.333333333333332 13.666666666666666 12.083333333333334 9.916666666666666
Cluster 59 15.857142857142858 12.761904761904763 13.523809523809524 12.476190476190476 14.285714285714286 15.476190476190476 12.238095238095237
Cluster 60 13.076923076923077 11.692307692307692 15.923076923076923 16.076923076923077 12.384615384615385 14.307692307692308 13.0
Cluster 61 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 62 14.23076923076923 10.615384615384615 14.692307692307692 17.53846153846154 13.538461538461538 15.76923076923077 10.153846153846153
Cluster 63 15.857142857142858 13.714285714285714 13.785714285714286 13.071428571428571 15.714285714285714 10.428571428571429 13.571428571428571
Cluster 64 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 65 18.0 8.714285714285714 15.857142857142858 17.714285714285715 13.142857142857142 9.142857142857142 14.428571428571429
Cluster 66 21.333333333333332 12.0 17.666666666666668 9.666666666666666 9.666666666666666 12.666666666666666 14.0
Cluster 67 17.818181818181817 12.727272727272727 15.727272727272727 13.454545454545455 12.363636363636363 14.272727272727273 10.272727272727273
Cluster 68 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 69 15.0 10.7 14.8 14.8 12.7 16.4 11.7
Cluster 70 17.333333333333332 10.166666666666666 11.25 19.0

12.916666666666666 14.0 11.916666666666666
Cluster 71 11.8 14.2 14.8 17.0 14.6 9.6 14.0
Cluster 72 15.272727272727273 17.727272727272727 13.0 11.818181818181818
12.090909090909092 16.09090909090909 10.454545454545455
Cluster 73 0.0 8.0 8.0 0.0 0.0 41.0 41.0
Cluster 74 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 75 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 76 9.428571428571429 14.5 13.785714285714286 17.214285714285715
13.714285714285714 12.642857142857142 14.857142857142858
Cluster 77 14.181818181818182 13.909090909090908 10.363636363636363
18.0
13.0 13.818181818181818 13.272727272727273
Cluster 78 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 79 0.0 50.0 50.0 0.0 0.0 0.0 0.0
Cluster 80 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 81 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 82 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 83 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 84 20.285714285714285 12.714285714285714 11.714285714285714
14.714285714285714 12.0 10.428571428571429 14.714285714285714
Cluster 85 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 86 15.285714285714286 14.285714285714286 11.571428571428571
16.285714285714285 9.571428571428571 15.714285714285714
13.857142857142858
Cluster 87 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 88 17.5 10.833333333333334 15.166666666666666 11.333333333333334
11.666666666666666 11.333333333333334 19.0
Cluster 89 18.153846153846153 16.692307692307693 11.615384615384615
12.538461538461538 13.538461538461538 12.153846153846153
11.76923076923077
Cluster 90 16.263157894736842 13.789473684210526 14.473684210526315
14.421052631578947 12.368421052631579 13.421052631578947
12.052631578947368
Cluster 91 21.0 10.0 10.0 10.0 9.0 18.0 18.0
Cluster 92 11.666666666666666 12.916666666666666 16.0 12.25 12.0
17.083333333333332 14.583333333333334
Cluster 93 12.294117647058824 14.764705882352942 13.941176470588236
16.41176470588235 13.0 14.470588235294118 11.470588235294118
Cluster 94 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 95 16.857142857142858 10.428571428571429 15.428571428571429

11.4285714285714129 15.857142857142858 11.571428571428571
14.571428571428571
Cluster 96 12.0 12.833333333333334 10.666666666666666 15.5 14.5
12.666666666666666 18.0
Cluster 97 15.4 11.533333333333333 13.533333333333333 14.0
11.933333333333334 16.8 13.466666666666667
Cluster 98 11.2 14.6 14.2 13.666666666666666 15.133333333333333
13.8 13.466666666666667
Cluster 99 14.083333333333334 11.0 16.5 13.416666666666666
12.916666666666666 14.5 14.083333333333334

0 14 (1%) 1 9 (1%) 2 1 (0%) 3 18 (2%) 4 11 (1%) 5 324 (32%) 6 10 (1%) 7 3 (0%) 8 5 (0%) 9 15 (1%) 10 22 (2%) 12 9 (1%) 14 11 (1%) 15 7 (1%) 16 18 (2%) 17 4 (0%) 18 10 (1%) 19 14 (1%) 26 3 (0%) 27 7 (1%) 28 7 (1%) 30 6 (1%) 32 3 (0%) 33 13 (1%) 34 10 (1%) 36 11 (1%) 37 14 (1%) 39 16 (2%) 40 21 (2%) 41 9 (1%) 43 15 (1%) 44 14 (1%) 45 6 (1%) 46 16 (2%) 53 9 (1%) 55 7 (1%) 56 11 (1%) 57 5 (0%) 58 12 (1%) 59 21 (2%) 60 13 (1%) 62 13 (1%) 63 14 (1%) 65 7 (1%) 66 3 (0%) 67 11 (1%) 69 10 (1%) 70 12 (1%) 71 5 (0%) 72 11 (1%) 73 1 (0%) 76 14 (1%) 77 11 (1%) 79 1 (0%) 84 7 (1%) 86 7 (1%) 88 6 (1%) 89 13 (1%) 90 19 (2%) 91 1 (0%) 92 12 (1%) 93 17 (2%) 95 7 (1%) 96 6 (1%) 97 15 (1%) 98 15 (1%) 99 12 (1%)

Scheme: weka.clusterers.SimpleKMeans -N 200 -S 1
Cluster centroids:
Cluster 0 16.25 9.75 13.75 12.5 13.75 18.75 11.75
Cluster 1 15.333333333333334 11.333333333333334 17.333333333333332
15.333333333333334 13.333333333333334 13.666666666666666
10.333333333333334
Cluster 2 12.833333333333334 9.166666666666666 12.333333333333334
16.166666666666668 14.333333333333334 17.166666666666668
14.333333333333334
Cluster 3 0.0 0.0 0.0 0.0 100.0 0.0 0.0
Cluster 4 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 5 13.25 12.25 16.25 15.25 9.75 16.75 12.75
Cluster 6 11.25 13.375 13.25 14.125 15.125 13.0 15.75
Cluster 7 11.5 13.75 15.375 14.625 13.25 15.25 12.25
Cluster 8 15.333333333333334 11.333333333333334 14.222222222222221
14.11111111111111 16.0 11.333333333333334 13.88888888888889

Cluster 9 13.666666666666666 15.0 13.333333333333334 15.0
12.666666666666666 11.333333333333334 14.666666666666666
Cluster 10 0.0 50.0 50.0 0.0 0.0 0.0 0.0
Cluster 11 17.666666666666668 7.0 13.333333333333334
18.666666666666668 13.666666666666666 13.666666666666666
12.333333333333334
Cluster 12 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 13 15.333333333333334 10.666666666666666 13.444444444444445
14.555555555555555 11.88888888888889 16.666666666666668
14.444444444444445
Cluster 14 14.333333333333334 11.666666666666666 16.0 11.666666666666666
14.666666666666666 12.666666666666666 16.0
Cluster 15 19.166666666666668 12.166666666666666 14.5 13.5
11.666666666666666 16.5 9.5
Cluster 16 16.428571428571427 12.571428571428571 11.142857142857142
18.714285714285715 13.571428571428571 13.142857142857142
11.0
Cluster 17 13.428571428571429 14.142857142857142 14.0 15.142857142857142
12.857142857142858 14.857142857142858 12.0
Cluster 18 22.0 13.0 17.5 10.0 10.0 12.5 12.5
Cluster 19 18.833333333333332 10.833333333333334 17.666666666666668
13.166666666666666 12.333333333333334 12.0 12.166666666666666
Cluster 20 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 21 15.5 10.5 14.666666666666666 17.833333333333332
11.333333333333334 13.833333333333334 12.333333333333334
Cluster 22 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 23 14.428571428571429 14.285714285714286 11.714285714285714
14.0 11.142857142857142 13.0 18.0
Cluster 24 15.5 10.0 13.5 18.0 13.0 17.25 10.25
Cluster 25 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 26 12.333333333333334 12.333333333333334 10.0 16.666666666666668
15.666666666666666 11.333333333333334 17.666666666666668
Cluster 27 10.0 11.0 20.0 13.0 11.0 16.0 15.0
Cluster 28 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 29 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 30 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 31 17.75 10.0 16.25 11.25 12.5 10.5 18.75
Cluster 32 14.285714285714286 13.428571428571429 10.0 17.714285714285715
12.714285714285714 14.0 14.285714285714286
Cluster 33 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Cluster 34 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 35 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 36 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 37 14.5 9.8 13.4 21.3 12.9 11.1 13.3
Cluster 38 12.0 13.0 20.0 7.0 13.0 13.0 18.0
Cluster 39 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 40 19.4 10.8 11.2 16.2 11.4 16.0 12.0
Cluster 41 11.666666666666666 16.666666666666668 20.333333333333332
10.333333333333334 11.666666666666666 14.0 12.333333333333334
Cluster 42 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 43 12.0 11.375 13.0 15.0 12.375 16.125 16.375
Cluster 44 20.25 10.75 13.25 8.75 12.25 14.75 16.0
Cluster 45 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 46 15.666666666666666 17.0 9.333333333333334 10.0 14.0 13.0
17.666666666666668
Cluster 47 13.857142857142858 11.714285714285714 18.0 13.142857142857142
11.285714285714286 14.428571428571429 14.142857142857142
Cluster 48 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 49 15.357142857142858 12.428571428571429 13.571428571428571
14.5
14.428571428571429 15.214285714285714 10.857142857142858
Cluster 50 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 51 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 52 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 53 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 54 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 55 12.75 13.5 15.0 14.75 13.5 12.5 14.75
Cluster 56 16.0 16.4 13.2 16.0 12.4 12.4 10.2
Cluster 57 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 58 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 59 16.833333333333332 10.833333333333334 10.5 16.0 12.0
17.666666666666668 12.333333333333334
Cluster 60 18.5 17.25 10.5 14.5 13.75 10.25 11.0
Cluster 61 16.4 10.6 14.2 12.2 12.6 17.0 13.8
Cluster 62 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 63 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 64 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 65 12.0 16.666666666666668 12.5 16.333333333333332 13.5
14.5 11.0
Cluster 66 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Cluster 67 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 68 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 69 13.0 15.571428571428571 10.714285714285714 14.571428571428571 14.714285714285714 16.142857142857142 11.714285714285714
Cluster 70 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 71 12.857142857142858 13.571428571428571 12.0 15.571428571428571 12.714285714285714 14.142857142857142 15.142857142857142
Cluster 72 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 73 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 74 14.333333333333334 14.88888888888889 12.666666666666666 17.0
14.666666666666666 12.555555555555555 10.11111111111111
Cluster 75 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 76 16.0 16.0 12.666666666666666 12.333333333333334 15.0 14.666666666666666 10.0
Cluster 77 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 78 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 79 14.0 14.75 9.75 10.5 12.5 19.0 15.25
Cluster 80 8.833333333333334 11.833333333333334 13.166666666666666 14.333333333333334 17.166666666666668 17.0 14.333333333333334
Cluster 81 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 82 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 83 12.0 14.4 11.0 21.2 12.4 12.6 12.8
Cluster 84 17.375 10.875 14.5 13.625 12.5 14.25 13.5
Cluster 85 12.9 14.4 14.4 12.9 15.0 14.2 12.6
Cluster 86 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 87 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 88 12.666666666666666 10.333333333333334 17.333333333333332 12.333333333333334 16.0 12.0 15.333333333333334
Cluster 89 16.0 13.0 19.0 12.0 12.0 9.0 15.0
Cluster 90 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 91 13.666666666666666 14.333333333333334 14.833333333333334 13.166666666666666 15.666666666666666 15.166666666666666
9.333333333333334
Cluster 92 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 93 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 94 20.5 12.833333333333334 14.0 12.333333333333334 12.666666666666666 11.666666666666666 12.5
Cluster 95 14.384615384615385 14.461538461538462 12.538461538461538 15.923076923076923 12.384615384615385 13.538461538461538

13.153846153846153
Cluster 96 13.857142857142858 12.428571428571429 12.428571428571429 18.285714285714285 14.714285714285714 13.285714285714286 11.142857142857142
Cluster 97 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 98 15.0 18.333333333333332 9.666666666666666 12.333333333333334 15.333333333333334 15.333333333333334 9.666666666666666
Cluster 99 14.0 12.625 15.5 16.875 12.125 13.25 12.375
Cluster 100 20.0 8.0 16.0 15.0 8.0 18.0 13.0
Cluster 101 13.833333333333334 12.5 14.0 12.5 14.166666666666666 18.5
11.333333333333334
Cluster 102 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 103 17.818181818181817 13.818181818181818 11.090909090909092 11.363636363636363 12.727272727272727 14.090909090909092 15.636363636363637
Cluster 104 19.666666666666668 13.166666666666666 10.0 12.0 13.5 15.666666666666666 12.333333333333334
Cluster 105 14.285714285714286 14.285714285714286 14.571428571428571 14.0 10.857142857142858 15.285714285714286 13.0
Cluster 106 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 107 15.6 13.6 13.0 11.4 14.2 13.2 15.0
Cluster 108 11.833333333333334 17.166666666666668 13.0 13.833333333333334 13.333333333333334 12.833333333333334 14.666666666666666
Cluster 109 12.2 16.8 12.2 12.0 14.2 16.6 12.0
Cluster 110 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 111 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 112 12.857142857142858 11.714285714285714 15.142857142857142 16.857142857142858 11.714285714285714 12.714285714285714
16.0
Cluster 113 14.8 10.2 14.8 10.8 13.8 17.4 14.8
Cluster 114 11.0 20.0 16.0 12.0 7.0 11.0 20.0
Cluster 115 16.0 11.333333333333334 18.0 16.0 11.333333333333334 11.0 13.333333333333334
Cluster 116 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 117 14.0 21.0 14.0 12.0 9.0 17.0 10.0
Cluster 118 11.5 12.0 14.5 12.833333333333334 13.0 18.833333333333332 13.666666666666666
Cluster 119 15.0 12.75 19.25 13.5 7.75 16.0 12.25
Cluster 120 12.25 12.25 16.5 11.0 14.75 17.0 12.75

Cluster 121 16.571428571428573 12.428571428571429 14.714285714285714
15.285714285714286 11.142857142857142 14.714285714285714
12.285714285714286
Cluster 122 9.666666666666666 10.0 11.666666666666666 24.333333333333332
12.0 10.333333333333334 18.0
Cluster 123 14.0 10.5 16.0 17.0 15.0 13.5 9.5
Cluster 124 14.0 16.0 15.0 15.0 17.0 12.0 7.0
Cluster 125 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 126 15.833333333333334 14.333333333333334 15.166666666666666
13.166666666666666 12.5 11.0 14.5
Cluster 127 15.8 17.0 14.0 14.8 10.4 14.4 10.2
Cluster 128 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 129 16.5 8.5 15.0 11.25 17.0 11.75 16.25
Cluster 130 16.75 13.75 16.25 14.75 13.0 12.25 10.0
Cluster 131 12.333333333333334 14.166666666666666 11.666666666666666
11.5 13.333333333333334 15.333333333333334 18.166666666666668
Cluster 132 10.25 15.5 16.375 12.5 14.125 14.125 13.5
Cluster 133 10.333333333333334 15.0 19.333333333333332 12.666666666666666
15.666666666666666 10.333333333333334 12.666666666666666
Cluster 134 15.285714285714286 9.857142857142858 15.0 14.857142857142858
12.714285714285714 15.857142857142858 12.428571428571429
Cluster 135 9.5 16.5 14.25 17.75 13.0 11.75 13.75
Cluster 136 13.0 15.0 17.0 9.0 13.0 20.0 10.0
Cluster 137 15.75 11.0 9.75 13.75 15.5 14.25 16.75
Cluster 138 12.0 12.0 15.0 17.0 18.0 10.0 12.0
Cluster 139 16.25 14.5 13.0 9.0 14.5 14.75 14.5
Cluster 140 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 141 18.571428571428573 13.857142857142858 11.0 15.142857142857142
9.428571428571429 12.428571428571429 16.0
Cluster 142 17.833333333333332 13.666666666666666 13.583333333333334
12.916666666666666 12.833333333333334 12.833333333333334 12.75
Cluster 143 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 144 10.75 8.25 12.0 18.5 13.5 16.0 17.5
Cluster 145 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 146 15.588235294117647 13.411764705882353 13.529411764705882
13.529411764705882 13.235294117647058 14.647058823529411
12.647058823529411
Cluster 147 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 148 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 149 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Cluster 150 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 151 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 152 14.0 13.666666666666666 15.666666666666666 16.333333333333332 11.333333333333334 17.666666666666668 8.0
Cluster 153 13.5 12.0 9.0 14.5 11.75 23.0 12.25
Cluster 154 15.555555555555555 14.777777777777779 15.222222222222221 10.88888888888889 12.444444444444445 14.333333333333334 12.666666666666666
Cluster 155 10.4 15.0 11.2 10.8 14.8 19.0 15.0
Cluster 156 12.857142857142858 12.142857142857142 10.0 13.857142857142858 14.285714285714286 18.285714285714285 15.142857142857142
Cluster 157 15.5 11.0 19.0 12.0 12.0 15.0 13.0
Cluster 158 22.0 16.0 14.0 6.0 14.0 17.0 8.0
Cluster 159 12.5 11.0 16.0 16.166666666666668 13.833333333333334 13.666666666666666 13.166666666666666
Cluster 160 12.571428571428571 11.571428571428571 14.0 16.857142857142858 14.0 16.857142857142858 11.0
Cluster 161 15.833333333333334 13.333333333333334 12.333333333333334 13.666666666666666 18.166666666666668 9.833333333333334 13.5
Cluster 162 7.5 19.0 11.0 15.5 15.0 10.5 17.5
Cluster 163 10.0 12.222222222222221 14.0 17.555555555555557 13.88888888888889 13.555555555555555 14.88888888888889
Cluster 164 14.846153846153847 10.76923076923077 12.153846153846153 16.46153846153846 13.615384615384615 12.615384615384615 15.923076923076923
Cluster 165 13.4 12.8 12.0 11.4 18.0 12.0 16.8
Cluster 166 17.333333333333332 16.166666666666668 13.5 11.0 14.833333333333334 11.666666666666666 11.833333333333334
Cluster 167 11.75 14.75 14.75 17.0 13.75 9.5 14.5
Cluster 168 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 169 16.4 12.266666666666667 11.933333333333334 15.8 12.733333333333333 14.0 13.266666666666667
Cluster 170 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 171 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 172 14.25 15.75 13.25 11.5 11.25 16.0 13.75
Cluster 173 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 174 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 175 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 176 11.0 20.5 15.0 10.0 12.5 18.0 9.0
Cluster 177 0.0 8.0 8.0 0.0 0.0 41.0 41.0

Cluster 178 11.333333333333334 13.333333333333334 16.666666666666668 12.333333333333334 13.666666666666666 9.333333333333334 19.333333333333332
Cluster 179 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 180 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 181 17.666666666666668 11.333333333333334 8.666666666666666 20.0 11.333333333333334 14.666666666666666 12.666666666666666
Cluster 182 15.5 14.666666666666666 15.833333333333334 11.166666666666666 11.666666666666666 17.166666666666668 10.833333333333334
Cluster 183 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 184 12.75 14.5 13.75 12.25 18.75 11.0 12.5
Cluster 185 13.88888888888889 11.222222222222221 13.333333333333334 17.11111111111111 12.777777777777779 15.222222222222221 12.88888888888889
Cluster 186 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 187 14.428571428571429 9.714285714285714 11.714285714285714 13.285714285714286 17.714285714285715 15.714285714285714 14.0
Cluster 188 16.333333333333332 9.0 19.666666666666668 17.666666666666668 14.333333333333334 9.0 10.333333333333334
Cluster 189 14.0 15.0 20.0 12.5 13.5 11.5 10.0
Cluster 190 12.666666666666666 14.333333333333334 14.333333333333334 12.666666666666666 11.0 16.0 15.333333333333334
Cluster 191 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 192 19.2 8.8 13.4 17.4 14.4 8.8 15.0
Cluster 193 15.5 13.5 11.5 16.25 8.75 17.0 14.25
Cluster 194 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 195 17.75 10.5 12.25 16.75 13.25 11.5 15.0
Cluster 196 14.0 20.0 10.0 13.0 8.0 16.0 15.0
Cluster 197 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Cluster 198 15.0 17.5 16.5 13.5 9.0 16.5 9.0
Cluster 199 13.833333333333334 11.583333333333334 13.916666666666666 14.0 14.666666666666666 13.916666666666666 14.583333333333334

0 4 (0%) 1 3 (0%) 2 6 (1%) 3 1 (0%) 4 324 (32%) 5 4 (0%) 6 8 (1%) 7 8 (1%) 8 9 (1%) 9 3 (0%) 10 1 (0%) 11 3 (0%) 13 9 (1%) 14 3 (0%) 15 6 (1%) 16 7 (1%) 17 7 (1%) 18 2 (0%) 19 6 (1%) 21 6 (1%) 23 7 (1%) 24 4 (0%) 26 3 (0%) 27 1 (0%) 31 4 (0%) 32 7 (1%) 37 10 (1%) 38 1 (0%) 40 5 (0%) 41 3 (0%) 43 8 (1%) 44 4 (0%) 46 3 (0%) 47 7 (1%) 49 14 (1%) 55 4 (0%) 56 5 (0%)

59 6 (1%) 60 4 (0%) 61 5 (0%) 65 6 (1%) 69 7 (1%) 71 7 (1%) 74 9 (1%) 76 3 (0%) 79 4 (0%) 80 6 (1%) 83 5 (0%) 84 8 (1%) 85 10 (1%) 88 3 (0%) 89 1 (0%) 91 6 (1%) 94 6 (1%) 95 13 (1%) 96 7 (1%) 98 3 (0%) 99 8 (1%) 100 1 (0%) 101 6 (1%) 103 11 (1%) 104 6 (1%) 105 7 (1%) 107 5 (0%) 108 6 (1%) 109 5 (0%) 112 7 (1%) 113 5 (0%) 114 1 (0%) 115 3 (0%) 117 1 (0%) 118 6 (1%) 119 4 (0%) 120 4 (0%) 121 7 (1%) 122 3 (0%) 123 2 (0%) 124 1 (0%) 126 6 (1%) 127 5 (0%) 129 4 (0%) 130 4 (0%) 131 6 (1%) 132 8 (1%) 133 3 (0%) 134 7 (1%) 135 4 (0%) 136 1 (0%) 137 4 (0%) 138 1 (0%) 139 4 (0%) 141 7 (1%) 142 12 (1%) 144 4 (0%) 146 17 (2%) 152 3 (0%) 153 4 (0%) 154 9 (1%) 155 5 (0%) 156 7 (1%) 157 2 (0%) 158 1 (0%) 159 6 (1%) 160 7 (1%) 161 6 (1%) 162 2 (0%) 163 9 (1%) 164 13 (1%) 165 5 (0%) 166 6 (1%) 167 4 (0%) 169 15 (1%) 172 4 (0%) 176 2 (0%) 177 1 (0%) 178 3 (0%) 181 3 (0%) 182 6 (1%) 184 4 (0%) 185 9 (1%) 187 7 (1%) 188 3 (0%) 189 2 (0%) 190 3 (0%) 192 5 (0%) 193 4 (0%) 195 4 (0%) 196 1 (0%) 198 2 (0%) 199 12 (1%)

Ergebnisse EM

In diesem Abschnitt sind sechs Testläufe der Analyse mit dem Algorithmus EM unter der WEKA-GUI aufgelistet. Wie bei dem Simple-K-Means sind die gewählten Parameter der Kopfzeile zu entnehmen (I=maximale Iteration, N=maximale Anzahl an Cluster, S=seed, M=minimale Standardabweichung). Ein Wert für N von -1 bedeutet dabei, dass die Anzahl von dem Algorithmus selbst bestimmt wird.

Scheme: weka.clusterers.EM -I 100 -N -1 -S 100 -M 1.0E-6

Number of clusters selected by cross validation: 4
Cluster: 0 Prior probability: 0.6733
Attribute: WG1 Normal Distribution. Mean = 14.6093 StdDev = 2.5476
Attribute: WG2 Normal Distribution. Mean = 12.9082 StdDev = 2.5135
Attribute: WG3 Normal Distribution. Mean = 13.5872 StdDev = 2.4866
Attribute: WG4 Normal Distribution. Mean = 14.4368 StdDev = 2.6176

Attribute: WG5 Normal Distribution. Mean = 13.2869 StdDev = 2.4161
Attribute: WG6 Normal Distribution. Mean = 14.2351 StdDev = 2.5436
Attribute: WG7 Normal Distribution. Mean = 13.3759 StdDev = 2.4151

Cluster: 1 Prior probability: 0.3227
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG2 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG3 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 0

Cluster: 2 Prior probability: 0.002
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG2 Normal Distribution. Mean = 29 StdDev = 21
Attribute: WG3 Normal Distribution. Mean = 29 StdDev = 21
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG6 Normal Distribution. Mean = 20.5 StdDev = 20.5
Attribute: WG7 Normal Distribution. Mean = 20.5 StdDev = 20.5

Cluster: 3 Prior probability: 0.002
Attribute: WG1 Normal Distribution. Mean = 4.466 StdDev = 4.5118
Attribute: WG2 Normal Distribution. Mean = 3.9638 StdDev = 4.0006
Attribute: WG3 Normal Distribution. Mean = 5.4516 StdDev = 5.5017
Attribute: WG4 Normal Distribution. Mean = 13.8627 StdDev =13.9928
Attribute: WG5 Normal Distribution. Mean = 56.4028 StdDev = 43.9939
Attribute: WG6 Normal Distribution. Mean = 4.9538 StdDev = 5.0007
Attribute: WG7 Normal Distribution. Mean = 8.918 StdDev = 8.9995
0 676 (67%) 1 324 (32%) 2 2 (0%) 3 2 (0%)
Log likelihood: 17.45413

Scheme: weka.clusterers.EM -I 30 -N -1 -S 1 -M 1.0E-6
Number of clusters selected by cross validation: 7
Cluster: 0 Prior probability: 0

Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG2 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG3 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 0

Cluster: 1 Prior probability: 0.0733
Attribute: WG1 Normal Distribution. Mean = 14.1229 StdDev = 3.2596
Attribute: WG2 Normal Distribution. Mean = 12.2097 StdDev = 3.1243
Attribute: WG3 Normal Distribution. Mean = 13.8307 StdDev = 2.9102
Attribute: WG4 Normal Distribution. Mean = 17.4056 StdDev = 3.4673
Attribute: WG5 Normal Distribution. Mean = 12.9473 StdDev = 1.792
Attribute: WG6 Normal Distribution. Mean = 11.5777 StdDev = 1.6909
Attribute: WG7 Normal Distribution. Mean = 14.304 StdDev = 2.968

Cluster: 2 Prior probability: 0.3171
Attribute: WG1 Normal Distribution. Mean = 14.4861 StdDev = 1.8761
Attribute: WG2 Normal Distribution. Mean = 12.899 StdDev = 1.9388
Attribute: WG3 Normal Distribution. Mean = 13.8354 StdDev = 1.68
Attribute: WG4 Normal Distribution. Mean = 14.9046 StdDev = 1.8172
Attribute: WG5 Normal Distribution. Mean = 13.1629 StdDev = 1.9567
Attribute: WG6 Normal Distribution. Mean = 14.3372 StdDev = 1.7567
Attribute: WG7 Normal Distribution. Mean = 12.8178 StdDev = 1.8615

Cluster: 3 Prior probability: 0.3227
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 0

Attribute: WG2 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG3 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 0

Cluster: 4 Prior probability: 0.1883
Attribute: WG1 Normal Distribution. Mean = 14.9725 StdDev = 3.0384
Attribute: WG2 Normal Distribution. Mean = 13.9526 StdDev = 2.8308
Attribute: WG3 Normal Distribution. Mean = 14.1814 StdDev = 3.0391
Attribute: WG4 Normal Distribution. Mean = 12.1571 StdDev = 1.797
Attribute: WG5 Normal Distribution. Mean = 13.1428 StdDev = 2.8964
Attribute: WG6 Normal Distribution. Mean = 14.545 StdDev = 2.7335
Attribute: WG7 Normal Distribution. Mean = 13.489 StdDev = 2.8439

Cluster: 5 Prior probability: 0.003
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG2 Normal Distribution. Mean = 19.3333 StdDev = 21.9292
Attribute: WG3 Normal Distribution. Mean = 19.3333 StdDev = 21.9292
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG5 Normal Distribution. Mean = 33.3333 StdDev = 47.1405
Attribute: WG6 Normal Distribution. Mean = 13.6667 StdDev = 19.3276
Attribute: WG7 Normal Distribution. Mean = 13.6667 StdDev = 19.3276

Cluster: 6 Prior probability: 0.0955
Attribute: WG1 Normal Distribution. Mean = 14.6259 StdDev = 2.7707
Attribute: WG2 Normal Distribution. Mean = 11.3634 StdDev = 2.0527

Attribute: WG3 Normal Distribution. Mean = 11.3579 StdDev = 1.9557
Attribute: WG4 Normal Distribution. Mean = 15.2066 StdDev = 2.328
Attribute: WG5 Normal Distribution. Mean = 14.2408 StdDev = 2.9122
Attribute: WG6 Normal Distribution. Mean = 15.2886 StdDev = 3.4706
Attribute: WG7 Normal Distribution. Mean = 14.3688 StdDev = 2.1227
1 56 (6%) 2 349 (35%) 3 324 (32%) 4 192 (19%) 5 3 (0%) 6 80 (8%)
Log likelihood: 17.59983

Scheme: weka.clusterers.EM -I 20 -N -1 -S 12 -M 1.0E-6
Number of clusters selected by cross validation: 4
Cluster: 0 Prior probability: 0.6732
Attribute: WG1 Normal Distribution. Mean = 14.6095 StdDev = 2.5478
Attribute: WG2 Normal Distribution. Mean = 12.9085 StdDev = 2.5136
Attribute: WG3 Normal Distribution. Mean = 13.5882 StdDev = 2.4857
Attribute: WG4 Normal Distribution. Mean = 14.4372 StdDev = 2.6181
Attribute: WG5 Normal Distribution. Mean = 13.2875 StdDev = 2.4159
Attribute: WG6 Normal Distribution. Mean = 14.2328 StdDev = 2.5384
Attribute: WG7 Normal Distribution. Mean = 13.3756 StdDev = 2.4152

Cluster: 1 Prior probability: 0.3227
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG2 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG3 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 0

Cluster: 2 Prior probability: 0.0031
Attribute: WG1 Normal Distribution. Mean = 3.3347 StdDev = 4.5708
Attribute: WG2 Normal Distribution. Mean = 5.5256 StdDev = 3.8615
Attribute: WG3 Normal Distribution. Mean = 6.3595 StdDev = 4.5647
Attribute: WG4 Normal Distribution. Mean = 9.3185 StdDev = 12.9028
Attribute: WG5 Normal Distribution. Mean = 36.499 StdDev = 44.1584
Attribute: WG6 Normal Distribution. Mean = 17.438 StdDev = 17.2807
Attribute: WG7 Normal Distribution. Mean = 19.5034 StdDev =16.5479

Cluster: 3 Prior probability: 0.001
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG2 Normal Distribution. Mean = 50 StdDev = 0
Attribute: WG3 Normal Distribution. Mean = 50 StdDev = 0
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 0
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 0

0 676 (67%) 1 324 (32%) 2 3 (0%) 3 1 (0%)
Log likelihood: 17.47379

Scheme: weka.clusterers.EM -I 20 -N -1 -S 12 -M 2.0
Number of clusters selected by cross validation: 7
Cluster: 0 Prior probability: 0.001
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG2 Normal Distribution. Mean = 50 StdDev = 2
Attribute: WG3 Normal Distribution. Mean = 50 StdDev = 2
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 2

Cluster: 1 Prior probability: 0
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG2 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG3 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 2

Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 2

Cluster: 2 Prior probability: 0.2707
Attribute: WG1 Normal Distribution. Mean = 14.6153 StdDev = 3.1969
Attribute: WG2 Normal Distribution. Mean = 12.8754 StdDev = 3.1509
Attribute: WG3 Normal Distribution. Mean = 13.2653 StdDev = 3.1223
Attribute: WG4 Normal Distribution. Mean = 14.3318 StdDev = 3.4063
Attribute: WG5 Normal Distribution. Mean = 13.4042 StdDev = 2.8482
Attribute: WG6 Normal Distribution. Mean = 13.9879 StdDev = 3.2157
Attribute: WG7 Normal Distribution. Mean = 13.9435 StdDev = 2.8931

Cluster: 3 Prior probability: 0.4036
Attribute: WG1 Normal Distribution. Mean = 14.5917 StdDev = 2.0097
Attribute: WG2 Normal Distribution. Mean = 12.9183 StdDev = 2
Attribute: WG3 Normal Distribution. Mean = 13.7976 StdDev = 2
Attribute: WG4 Normal Distribution. Mean = 14.5403 StdDev = 2.0159
Attribute: WG5 Normal Distribution. Mean = 13.2049 StdDev = 2.0688
Attribute: WG6 Normal Distribution. Mean = 14.3912 StdDev = 2
Attribute: WG7 Normal Distribution. Mean = 13.0049 StdDev = 2

Cluster: 4 Prior probability: 0.3227
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG2 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG3 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 2

Cluster: 5 Prior probability: 0.001

Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG2 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG3 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG5 Normal Distribution. Mean = 100 StdDev = 2
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 2

Cluster: 6 Prior probability: 0.001
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG2 Normal Distribution. Mean = 8 StdDev = 2
Attribute: WG3 Normal Distribution. Mean = 8 StdDev = 2
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG6 Normal Distribution. Mean = 41 StdDev = 2
Attribute: WG7 Normal Distribution. Mean = 41 StdDev = 2

0 1 (0%) 2 226 (23%) 3 451 (45%) 4 324 (32%) 5 1 (0%) 6 1 (0%)
Log likelihood: -15.29293

Scheme: weka.clusterers.EM -I 20 -N 6 -S 12 -M 2.0
Number of clusters: 6
Cluster: 0 Prior probability: 0.3227
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG2 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG3 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 2

Cluster: 1 Prior probability: 0.1087
Attribute: WG1 Normal Distribution. Mean = 15.1518 StdDev = 3.5092
Attribute: WG2 Normal Distribution. Mean = 12.1836 StdDev = 3.0847
Attribute: WG3 Normal Distribution. Mean = 13.6184 StdDev = 3.1245
Attribute: WG4 Normal Distribution. Mean = 15.842 StdDev = 4.0059

Attribute: WG5 Normal Distribution. Mean = 12.7998 StdDev = 2.285
Attribute: WG6 Normal Distribution. Mean = 12.4268 StdDev = 2.467
Attribute: WG7 Normal Distribution. Mean = 14.435 StdDev = 3.1475

Cluster: 2 Prior probability: 0.179
Attribute: WG1 Normal Distribution. Mean = 14.256 StdDev = 2.8522
Attribute: WG2 Normal Distribution. Mean = 13.6067 StdDev = 2.9586
Attribute: WG3 Normal Distribution. Mean = 13.3315 StdDev = 3.0696
Attribute: WG4 Normal Distribution. Mean = 12.6677 StdDev = 2
Attribute: WG5 Normal Distribution. Mean = 13.8815 StdDev = 3.0403
Attribute: WG6 Normal Distribution. Mean = 15.1013 StdDev = 3.1475
Attribute: WG7 Normal Distribution. Mean = 13.5581 StdDev = 2.586

Cluster: 3 Prior probability: 0.3866
Attribute: WG1 Normal Distribution. Mean = 14.6034 StdDev = 2.0004
Attribute: WG2 Normal Distribution. Mean = 12.7805 StdDev = 2
Attribute: WG3 Normal Distribution. Mean = 13.6893 StdDev = 2
Attribute: WG4 Normal Distribution. Mean = 14.8848 StdDev = 2
Attribute: WG5 Normal Distribution. Mean = 13.1489 StdDev = 2.0465
Attribute: WG6 Normal Distribution. Mean = 14.34 StdDev = 2
Attribute: WG7 Normal Distribution. Mean = 13.0029 StdDev = 2

Cluster: 4 Prior probability: 0.002
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG2 Normal Distribution. Mean = 4 StdDev = 4
Attribute: WG3 Normal Distribution. Mean = 4 StdDev = 4
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG5 Normal Distribution. Mean = 49.9996 StdDev = 50
Attribute: WG6 Normal Distribution. Mean = 20.4998 StdDev = 20.5
Attribute: WG7 Normal Distribution. Mean = 20.4998 StdDev = 20.5

Cluster: 5 Prior probability: 0.001
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG2 Normal Distribution. Mean = 50 StdDev = 2
Attribute: WG3 Normal Distribution. Mean = 50 StdDev = 2
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 2
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 2

0 324 (32%) 1 81 (8%) 2 156 (16%) 3 440 (44%) 4 2 (0%) 5 1 (0%)

Scheme: weka.clusterers.EM -I 20 -N 12 -S 1 -M 0.0010
Number of clusters: 12
Cluster: 0 Prior probability: 0.2057
Attribute: WG1 Normal Distribution. Mean = 14.6384 StdDev = 1.7828
Attribute: WG2 Normal Distribution. Mean = 13.1461 StdDev = 1.7971
Attribute: WG3 Normal Distribution. Mean = 14.0853 StdDev = 1.5509
Attribute: WG4 Normal Distribution. Mean = 14.7133 StdDev = 1.81
Attribute: WG5 Normal Distribution. Mean = 13.0497 StdDev = 1.7652
Attribute: WG6 Normal Distribution. Mean = 14.3205 StdDev = 1.6025
Attribute: WG7 Normal Distribution. Mean = 12.4908 StdDev = 1.6757

Cluster: 1 Prior probability: 0.001
Attribute: WG1 Normal Distribution. Mean = 14 StdDev = 0.001
Attribute: WG2 Normal Distribution. Mean = 12 StdDev = 0.001
Attribute: WG3 Normal Distribution. Mean = 8 StdDev = 0.001
Attribute: WG4 Normal Distribution. Mean = 17 StdDev = 0.001
Attribute: WG5 Normal Distribution. Mean = 11 StdDev = 0.001
Attribute: WG6 Normal Distribution. Mean = 22 StdDev = 0.001
Attribute: WG7 Normal Distribution. Mean = 12 StdDev = 0.001

Cluster: 2 Prior probability: 0.3227

Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG2 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG3 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 0.001

Cluster: 3 Prior probability: 0.001
Attribute: WG1 Normal Distribution. Mean = 13 StdDev = 0.001
Attribute: WG2 Normal Distribution. Mean = 11 StdDev = 0.001
Attribute: WG3 Normal Distribution. Mean = 8 StdDev = 0.001
Attribute: WG4 Normal Distribution. Mean = 13 StdDev = 0.001
Attribute: WG5 Normal Distribution. Mean = 10 StdDev = 0.001
Attribute: WG6 Normal Distribution. Mean = 27 StdDev = 0.001
Attribute: WG7 Normal Distribution. Mean = 15 StdDev = 0.001

Cluster: 4 Prior probability: 0.0385
Attribute: WG1 Normal Distribution. Mean = 11.9821 StdDev = 2.3973
Attribute: WG2 Normal Distribution. Mean = 12.7893 StdDev = 3.3158
Attribute: WG3 Normal Distribution. Mean = 12.5828 StdDev = 2.0101
Attribute: WG4 Normal Distribution. Mean = 18.875 StdDev = 2.9858
Attribute: WG5 Normal Distribution. Mean = 13.1322 StdDev = 1.2565
Attribute: WG6 Normal Distribution. Mean = 11.9132 StdDev = 1.6342
Attribute: WG7 Normal Distribution. Mean = 15.0775 StdDev = 2.4327

Cluster: 5 Prior probability: 0.001
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG2 Normal Distribution. Mean = 50 StdDev = 0.001
Attribute: WG3 Normal Distribution. Mean = 50 StdDev = 0.001
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 0.001

Cluster: 6 Prior probability: 0.1166
Attribute: WG1 Normal Distribution. Mean = 13.4269 StdDev = 2.3363
Attribute: WG2 Normal Distribution. Mean = 12.9301 StdDev = 2.3039
Attribute: WG3 Normal Distribution. Mean = 13.0952 StdDev = 2.6215
Attribute: WG4 Normal Distribution. Mean = 12.876 StdDev = 1.6419
Attribute: WG5 Normal Distribution. Mean = 15.0636 StdDev = 2.4424
Attribute: WG6 Normal Distribution. Mean = 15.0745 StdDev = 3.0944
Attribute: WG7 Normal Distribution. Mean = 13.8837 StdDev = 2.2132

Cluster: 7 Prior probability: 0.001
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG2 Normal Distribution. Mean = 8 StdDev = 0.001
Attribute: WG3 Normal Distribution. Mean = 8 StdDev = 0.001
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG5 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG6 Normal Distribution. Mean = 41 StdDev = 0.001
Attribute: WG7 Normal Distribution. Mean = 41 StdDev = 0.001

Cluster: 8 Prior probability: 0.0868
Attribute: WG1 Normal Distribution. Mean = 14.7589 StdDev = 2.968
Attribute: WG2 Normal Distribution. Mean = 15.4452 StdDev = 2.6317
Attribute: WG3 Normal Distribution. Mean = 14.4035 StdDev = 3.4632
Attribute: WG4 Normal Distribution. Mean = 12.1647 StdDev = 1.9872
Attribute: WG5 Normal Distribution. Mean = 11.9574 StdDev = 2.5487
Attribute: WG6 Normal Distribution. Mean = 14.5637 StdDev = 2.5965
Attribute: WG7 Normal Distribution. Mean = 13.1626 StdDev = 3.2557

Cluster: 9 Prior probability: 0.1528
Attribute: WG1 Normal Distribution. Mean = 14.6042 StdDev = 2.1634
Attribute: WG2 Normal Distribution. Mean = 12.0234 StdDev = 2.1443
Attribute: WG3 Normal Distribution. Mean = 12.8355 StdDev = 2.0442
Attribute: WG4 Normal Distribution. Mean = 15.4799 StdDev = 2.0215
Attribute: WG5 Normal Distribution. Mean = 13.2403 StdDev = 2.3675
Attribute: WG6 Normal Distribution. Mean = 14.4899 StdDev = 2.2566
Attribute: WG7 Normal Distribution. Mean = 13.7664 StdDev = 2.1684

Cluster: 10 Prior probability: 0.001
Attribute: WG1 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG2 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG3 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG4 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG5 Normal Distribution. Mean = 100 StdDev = 0.001
Attribute: WG6 Normal Distribution. Mean = 0 StdDev = 0.001
Attribute: WG7 Normal Distribution. Mean = 0 StdDev = 0.001

Cluster: 11 Prior probability: 0.0719
Attribute: WG1 Normal Distribution. Mean = 17.6367 StdDev = 2.1496
Attribute: WG2 Normal Distribution. Mean = 11.1332 StdDev = 2.2602
Attribute: WG3 Normal Distribution. Mean = 14.2594 StdDev = 3.1314
Attribute: WG4 Normal Distribution. Mean = 14.3869 StdDev = 3.3934
Attribute: WG5 Normal Distribution. Mean = 12.9098 StdDev = 2.8055
Attribute: WG6 Normal Distribution. Mean = 12.5936 StdDev = 2.7626
Attribute: WG7 Normal Distribution. Mean = 13.6695 StdDev =

2.9089
0 249 (25%) 1 1 (0%) 2 324 (32%) 3 1 (0%) 4 34 (3%) 5 1 (0%) 6 120 (12%) 7 1 (0%) 8 82 (8%) 9 128 (13%) 10 1 (0%) 11 62 (6%)
Log likelihood: 2.2629

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Ama03] Amazon GmbH, München. Amazon.de, **2003**. URL http://www.amazon.de/

[Dik03] Diko. Projektgruppe internetbasierter Handelsszenarien, Javadoc, **2003**. URL http://diko-project.de/dokumente/javadoc/index.html

[Fis87] D. H. Fisher. Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning*, **1987**

[iH03] Projektgruppe Personalisierung internetbasierter Handelsszenarien. Kartenanbieter. **2003**

[Inf88] Studienkommission Informatik. Projektgruppenpapier, **1988**. URL http://www-is.informatik.uni-oldenburg.de/~dibo/teaching/pg%_fb10/PG_Ordnung.html

[LNWJ] Yingjiu Li, Peng Ning, X. Sean Wang und Sushil Jajodia. Generating Market Basket Data with Temporal Information. URL citeseer.nj.nec.com/525901.html

[LNWJ01] Yingjiu Li, Peng Ning, Xiaoyang Sean Wang und Sushil Jajodia. Discovering Calendar-based Temporal Association Rules. In *TIME*, Seiten 111–118. **2001**. URL citeseer.nj.nec.com/li01discovering.html

[LWD98] F. Leisch, A. Weingessel und E. Dimitriadou. Competitive learning for binary valued data, **1998**. URL citeseer.nj.nec.com/article/leisch98competitive.html

[Mic03] Sun Microsystems. Java 2 Platform, Standard Edition, v 1.4.1 API Specification, **2003**. URL http://java.sun.com/j2se/1.4.1/docs/api/

[Obj01] OBJECT MANAGEMENT GROUP. Common Warehouse Metamodel (CWM) Specification. Version 1.0, **2001**

[PM00] DAN PELLEG und ANDREW MOORE. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proc. 17th International Conf. on Machine Learning*, Seiten 727–734. Morgan Kaufmann, San Francisco, CA, **2000**. URL `citeseer.nj.nec.com/pelleg00xmeans.html`

[Pre03] MATTHIAS PRETZER. Clustering und Klassifikation. **2003**. URL `http://diko-project.de/dokumente/ausarbeitungen/pretzer.ps.gz`

[Saa03] HELGE SAATHOFF. Assoziationsanalyse und Konzeptbeschreibung. Seiten 163–196, **2003**

[Sof03] SLEEPYCAT SOFTWARE. Berkeley DB, **2003**. URL `http://sleepycat.com`

[WF01] IAN H. WITTEN und EIBE FRANK. *Data Mining, Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, **2001**. ISBN 1-55860-552-5

[Wie03] OLIVER WIEN. Temporale Aspekte des Data Mining. **2003**. URL `http://www.diko-project.de/dokumente/ausarbeitungen/wien.pdf`

Index

Glossar

Aggregation: Die ∼ bezeichnet das Zusammenfassen von Daten mittels einer Berechnungsvorschrift

Aggregationsfunktion: Diese Berechnungsfunktion bildet eine Wertemenge auf einen einzelnen Wert ab

Analyse: Die ∼ bezeichnet alle Operationen, die mit den Daten eines ↑ Data Warehouse durchgeführt werden, um neue Informationen zu generieren

Arbeitsbereich: Der ∼ hält die Daten des ↑ Datenbeschaffungsbereichs

Arithmetisches Mittel: Das ∼ beschreibt den Wert, den man umgangssprachlich als Durchschnittswert bezeichnet. Er wird berechnet, in dem alle Werte aufsummiert werden und anschließend das Ergebnis durch die Anzahl der Werte geteilt wird

Attribut: Ein ∼ ist die allgemeine Bezeichnung für Eigenschaften von Datenobjekten wie z.B. Farbe und Größe. Im Zusammenhang mit relationalen Datenbanken werden die Elemente einer Spalte als Attributwerte bezeichnet

Attribut-Zeitstempelung: Jedes ↑ Attribut einer ↑ Relation erhält eine eigene Temporalisierung. Bei der ∼ erfolgt die Implementierung der ↑ Zeitstempel auf Attributebene

Ausreißer: ∼ sind Daten, die verglichen mit anderen Werten, sehr groß oder sehr klein sind

Average Linkage: Das ∼-Verfahren ist ein Verfahren der hierarchischen Clusteranalyse, wobei ∼ die durchschnittliche Ähnlichkeit aller Paare von Individuen x Element C und y Element D bezeichnet

Basisdatenbank: Die anwendungsneutrale ∼ stellt eine integrierte Datenbasis dar. Ihre Aufgabe ist die Sammlung, Integration und Verteilung der Daten

Baum: Eine dynamische Datenstruktur, die z.B. bei hierarchischen Beziehungen und bei rekursiven Strukturen Verwendung findet, wird als ∼ bezeichnet. Bäume sind (un)-gerichtete, zyklenfreie Graphen die einen Knoten besitzen, von dem aus jeder andere Knoten des Graphen auf genau einem Weg zu erreichen ist

Belegdaten: Mit ∼ werden alle Ein- und Ausgänge von Ware bezeichnet. Dieses beinhaltet auch die Bondaten, wobei es sich um einzelne Verkäufe handelt. Dabei werden sämtliche Positionen (Filiale, Kasse, Zeitpunkt, Warenwert, etc.) genau aufgeschlüsselt

Benutzerdefinierte Zeit: Bei der ∼ (engl. user-defined-time) handelt es sich um eine Domäne für zeitliche Werte wie z.B. Integer für Zahlenwerte. Für temporale DBMS hat diese Domäne keine spezielle Bedeutung

Binning: ∼ ist eine Methode, um Werte in Gruppen (sog. bins=Eimer) einzuteilen

Bitemporales Modell: Beim bitemporalen Modell werden sowohl ↑ Transaktions- als auch ↑ Gültigkeitszeit gespeichert

Bonität: Die ∼ beschreibt die relative Ertragskraft des Schuldners in der Zukunft

Bootstrapping: ∼ ist eine Technik, mit der man aus einer kleinen ↑ Stichprobe mit Hilfe von statistischen Methoden repräsentative Ergebnisse erzielen kann. Dazu werden n verschiedene Mengen durch Ziehen mit Zurücklegen aus der ↑ Stichprobe entnommen und das jeweilige Verfahren auf diesen zufällig erzeugten Mengen angewendet

Box-and-Whisker-Plot: ∼ bezeichnet die graphische Darstellung einer Variablenverteilung

Business Understanding: ∼ ist die erste Phase im ↑ CRISP-DM-Modell: Umfasst die Betrachtung des Geschäftsumfeldes, Festlegen von Zielen und Erfolgsfaktoren für das Projekt und für das ↑ Data Mining und die Bestimmung eines Projektplanes

C4.5: ∼ zählt zu den bekanntesten ↑ Entscheidungsbaumverfahren aus dem Bereich des induktiven Lernens. Das Auswahlkriterium in ∼ basiert auf informationstheoretischen Überlegungen, es wird ein error-based Pruning eingesetzt

Cache: Ein ∼ ist Zwischenspeicher. Im Zusammenhang mit ↑ Web Usage Mining ist damit der ∼ eines ↑ Browsers oder ↑ Proxy Servers gemeint, in dem Seiten aus dem Internet zwischengespeichert werden. Dies kann Performance Vorteile mit sich bringen, da nicht mehr jede Seite vom ↑ Webserver angefordert werden muss

CART: Die ∼ - Methode (Classification and ↑ Regression Trees) wurde von Breiman et al. 1984 als Ergebnis mehrjähriger Forschungsarbeit entwickelt. Das ∼ ↑ Entscheidungsbaumverfahren zählt zu den bekanntesten Top-Down Ansätzen mit entsprechender ↑ Pruning-Strategie

CHAID: Der ∼-Algorithmus stellt ein Segmentierungs-Verfahren dar, dessen Auswahlkriterium auf dem Chi-Quadrat-Test beruht. Er zählt zu den direkten Top-Down-Verfahren ohne die Verwendung einer nachfolgenden Pruning-Phase

Chronon: Ein ∼ ist die kleinste, gewählte Zeiteinheit innerhalb eines Datenbankmodells

Cluster: Ein ∼ ist eine Menge von Objekten, die zueinander bezüglich (eines Teils) ihrer ↑ Attribute eine hohe Ähnlichkeit und zu Objekten außerhalb des Clusters eine geringe Ähnlichkeit haben

Clustering: Beim ∼ werden ↑ Cluster gebildet, die bezüglich der zu analysierenden Daten in sich möglichst homogen und untereinander möglichst heterogen sind

Cookie: Im Internet-Umfeld eine kleine Textdatei, die lokal auf dem Rechner der surfenden Person abgelegt wird und in welcher Informa-

tionen abgespeichert werden, die im Zusammenhang mit der aktuellen ↑ Website stehen. Das ∼ wird bei jedem ↑ Request an die ↑ Website mitgeschickt, die das ∼ gesetzt hat. Somit können z.B. eindeutige Kennungen auf einem Client gespeichert werden. Der Einsatz erfolgt z.B. beim Warenkorb einer kommerziellen Site, zur Personalisierung einer ↑ Website oder für die Nutzererkennung

CRISP-DM: Der Cross-Industry Standard Process for Data Mining ist ein Vorgehensmodell für den ↑ KDD-Prozess. Entwickelt wurde der ∼ von der Statistical Package for the Social Sciences Incorporation (SPSS inc.), der National Cash Register Company (NCR) und Daimler-Chrysler

Customer Relationship Management(CRM): ∼ ist die Pflege von Kundenbeziehungen mittels einer Softwarelösung, die alle Geschäftsvorgänge und Informationen eines Kunden erfasst.(z.B. die ↑ Analyse aller Kauftransaktionen eines Kunden). Ziel ist eine weitreichende Integration von Kunden, Mitarbeitern, Prozessen und Systemen

Data cleaning: ↑ Datenbereinigung

Data integration: ↑ Datenintegration

Data Mart: Das ∼-Konzept liefert eine inhaltlich beschränkte Sicht auf ein ↑ Data Warehouse. Aus Datenbanksicht handelt es sich beim Data-Mart um eine Verteilung eines Data-Warehouse-Datenbestandes. Entweder sind die Data Marts abhängig als Extrakte aus dem integrierten Datenbestand der ↑ Basisdatenbank ohne Bereinigung/Normierung zu verstehen oder als unabhängige Data Marts als isolierte Sichten auf die Quellsysteme unter Nichtbeachtung der ↑ Basisdatenbank

Data Mining: ∼ bezeichnet eine Technik zur automatischen Entdeckung neuer, nicht-trivialer und voraussichtlich nützlicher Abhängigkeiten innerhalb großer oder komplexer Datenbestände. ∼ wird dabei als einer von mehreren Schritten im ↑ KDD-Prozess verstanden

Data Preparation: ↑ Datenvorverarbeitung

Data reduction: ↑ Datenreduktion

Data smoothing: ↑ Datenglättung

Data transformation: ↑ Datentransformation

Data Understanding: ∼ ist die zweite Phase im ↑ CRISP-DM; sie beginnt mit der Sammlung und der Beschäftigung mit den notwendigen Daten, um etwaige Probleme in Umfang oder Qualität herauszufiltern. Weiterhin sind in dieser Phase interessante Mengen zu finden, um Hypothesen für versteckte Informationen zu formulieren

Data Warehouse (DW): Alle für Analysezwecke relevanten Daten werden in einem großen Informationssystem, dem ∼, gespeichert, damit diese dann per ↑ Data Mining oder mit anderen Analysemethoden und -tools (z.B. ↑ OLAP) analysiert, aufbereitet und ausgewertet werden können Ein ∼ ist somit eine physische Datenbank, die eine integrierte Sicht auf aktuelle und historisierte Daten bietet

Data Warehousing: ∼ umfasst in einem dynamischem Prozess alle Schritte der Datenbeschaffung, des Speicherns und der ↑ Analyse. Es beinhaltet also die ↑ Datenintegration, ↑ Datentransformation, Datenkonsolidierung, ↑ Datenbereinigung und Speicherung der Daten sowie die Datenbereiststellung für analytische Zwecke und Interpretationen

Data Warehouse Manager: Der ∼ initiiert, steuert und überwacht die einzelnen Prozesse in allen Phasen

Datenanalyse: Die ∼ befasst sich mit ↑ Data Mining (Entdeckungs)- und Verifikationsverfahren, die häufig auf klassischen Verfahren aufbauen

Datenbereinigung: Die ∼ bezeichnet ein Verfahren, fehlende Werte, sog. ↑ noisy data und Inkonsistenzen aus einem Datenbestand zu entfernen

Datenbeschaffungsbereich: Der ∼ enthält alle Komponenten, die funktional zwischen den ↑ Datenquellen und der ↑ Basisdatenbank liegen

Datenglättung: Die ∼ bezeichnet ein Verfahren, Ausreißer und sog. ↑ „noisy data“ aus einem Datenbestand zu entfernen und die darin enthaltenen Daten einander anzugleichen

Datenintegration: Die ∼ bezeichnet das Zusammenfügen der Daten mehrerer heterogener Datensätze von verschiedenen Quellen zu einem schlüssigen, kohärenten und homogenen Datensatz

Datenkompression: ∼ ist ein Verfahren zur Verkleinerung eines Datenbestandes, wobei dies verlustfrei oder verlustbehaftet geschehen kann in Bezug auf den Informationsgehalt

Datennormalisierung: ↑ Normalisierung

Datenquelle: Die ∼ beeinflusst durch die Art der Datenspeicherung die Analysefähigkeit eines Data-Warehouse-System. Sie stellt einen Bestand von Rohdaten mit Inhalten für den Analysezweck dar

Datenreduktion: Die ∼ umfasst verschiedene Strategien zur Verkleinerung des Volumens von Daten wie z.B. ↑ Aggregation, ↑ Dimensionsreduktion, ↑ Datenkompression oder ↑ Numerische Datenreduktion

Datentransformation: ∼ bezeichnet Verfahren, die der Homogenisierung vormals heterogener Daten dienen, um Daten in eine für das ↑ Data Mining notwendige Form zu transformieren

Datenvorverarbeitung: Ziel dieser Vorbereitungsphase des ↑ KDD-Prozesses ist die Bereitstellung der zu analysierenden Datensätze für die gewünschten ↑ Analysen und Verfahren, wozu die Aufgaben des Transformieren, Bereinigen und der Selektion in nicht festgesetzter Reihenfolge mehrfach wiederholt werden. Die ∼ ist die zweite und arbeitsintensivste Phase des ganzen ↑ KDD-Prozesses und beansprucht teilweise mehr als die Hälfte der ganzen zur Verfügung stehenden Zeit

Datenwertskonflikt: Der ∼ stellt einen Zustand dar, in dem in ein und der selben Entität nicht zueinander kompatible Datenwerte enthalten sind

Datenwürfel: ↑ Würfel

Deployment: ∼ ist die letzte Phase des allgemeinen KDD-Modelles. Die endgültigen Ergebnisse der ↑ Datenanalyse werden abschließend im ∼ für die verschiedenen Adressaten aufbereitet

Detaildaten: ∼ bezeichnen Daten der niedrigsten Stufe einer ↑ Dimensionenhierarchie

Dimension: Eine ∼ ist eine qualifizierende Eigenschaft eines ↑ Fakts. Sie stellt einen Aspekt des Auswertungskontextes dar

Dimensionenhierarchie: Eine Menge aufeinander aufbauender ↑ Hierarchieebenen wird ∼ genannt

Dimensioneninstanz: Die ∼ ist die Menge aller ↑ Dimensionenhierarchien auf Pfaden im ↑ Klassifikationsschema

Dimensionsreduktion: Die ∼ ist ein Verfahren, um die Dimensionalität eines zu untersuchenden Systems zu reduzieren. Möglichkeiten bietet hier bspw. die ↑ Hauptkomponentenanlyse

Diskrete Daten: ∼ bezeichnen feste, numerische Werte, wie sie beispielsweise durch Zählungen entstehen

Diskretisierung: ∼ ist ein Verfahren, das die Anzahl der Werte für ein durchgängiges ↑ Attribut durch Aufteilung des Wertebereichs in einzelne Intervalle verringern kann

Diskriminanzanalyse: Die ∼ untersucht den jeweiligen Datenbestand auf solche ↑ Attribute, die einen hohen Erklärungsgrad für eine bereits vorgegebene ↑ Klassifikation besitzen. Ein Beispiel ist hier die Anwendung der ∼ als Frühwarnsystem bei der Bonitätsprüfung

Document Root: Das ∼ ist das Hauptverzeichnis eines ↑ Webservers. Wird dieses angefordert, liefert der ↑ Webserver in der Regel eine Standarddatei wie z.B. `index.html` aus

Drill-down: ∼ ist die Umkehrung einer ↑ roll-up-Operation

Electronic Procurement: Unter ~ wird die elektronische Beschaffung, das heißt der Einkauf von Waren und Dienstleistungen über das Internet verstanden

Entscheidungsbaumverfahren: Bei dem ~ werden Objekte, deren Klassenzuordnung bekannt ist, sukzessive mit Hilfe einzelner Merkmale in Gruppen aufgeteilt, die in sich homogen, aber voneinander möglichst unterschiedlich sind. Am Ende des Verfahrens entsteht ein ↑ Baum, aus dessen Verzweigungskriterien Regeln gebildet werden können, die dann auf nicht zugeordnete Objekte angewendet werden können

ETL-Prozess: ETL steht für „Extraction Transformation Label“. Der ~ bezeichnet den Prozess der Datenverarbeitung von den ↑ Datenquellen bis zum ↑ Data Warehouse

Evaluation: ~ ist die fünfte Phase im ↑ CRISP-DM; nach Abschluss der Mining-Phase sind die gefundenen Ergebnisse in Form von Modellen einer kritischen Betrachtung zu unterziehen. Bevor die Ergebnisse zum ↑ Deployment freigegeben werden, müssen die zu Beginn aufgestellten Geschäftsziele hinsichtlich ihrer Erfüllung betrachtet werden. Nebenbei werden auch die Modellierungsmethoden und die Schritte des gesamten Prozesses überprüft, um etwaige Verbesserungspotentiale zu erkennen. Im Anschluss an diese Prüfungen wird über den Prozessfortgang und den Grad der Nutzung der Ergebnisse entschieden

Extraktionskomponente: Die ~ dient der Übertragung der Daten aus einer ↑ Datenquelle in den ↑ Arbeitsbereich. Sie unterstützt zusätzlich die Auswahl der Quellen, die importiert werden sollen

Factoring: ~ ist ein Finanzierungsgeschäft, bei dem ein spezialisiertes Finanzierungsinstitut (Factor) von einem Verkäufer dessen Forderungen aus Warenlieferungen und Dienstleistungen laufend oder einmalig ankauft und die Verwaltung der Forderungen übernimmt

Fakt: Ein ~ ist ein Objekt, das quantifizierende und qualifizierende Merkmale besitzt. Die quantifizierbaren Eigenschaften beinhalten für die Organisation relevante Daten, die während einer ↑ Datenanalyse weitergehend untersucht werden können. Qualifizierende Ei-

genschaften dienen der näheren Beschreibung der quantifizierbaren Eigenschaften, wodurch diese eine Bedeutung erhalten

Fast Constellation Schema: Bei einem ∼ sind neben den ↑ Basisdaten ebenso verdichtete Daten vorhanden

Fehlerquadratsumme: Die ∼ ist die Summe der quadratischen Abweichungen der einzelnen Meßwerte von ihrem Gruppenmittelwert und stellt darum die Variation innerhalb der zu behandelnden Gruppe dar

Frequent itemset: ∼ bezeichnet eine Menge von Elementen, die häufig zusammen auftreten

Gain-Chart: Ein ∼ ist eine Möglichkeit, die Qualität eines Klassifikators darzustellen. Ein ∼ wird dabei durch zwei Kurven charakterisiert, die den Informationsgewinn auf den Lerndaten und den Evaluierungsdaten darstellen. Je größer der Abstand zwischen den beiden Kurven ist, um so geeigneter ist der Klassifikator

Galaxie: Eine ∼ ist ein Schema mit mehreren Fakttabellen

Gaußverteilung: Eine Gauß- oder auch ↑ Normalverteilung ist eine bestimmte Wahrscheinlichkeitsverteilung, die durch ihren Mittelwert und ihre ↑ Varianz eindeutig bestimmt ist

Generalisierung: ∼ bezeichnet eine Methode, die eine kompakte Beschreibung einer gegebenen Datenmenge zurückliefert. Wird in der attributorientierten Induktion und beim ↑ OLAP verwendet

Gläserner Kunde: Die Angst von Konsumenten vor dem Zustand, daß Unternehmen durch Datensammlung ein Persönlichkeitsprofil erstellen können und Kunden damit beeinflussen, wird in dem Begriff ∼ zusammengefasst

Gültigkeitszeit: ∼ ist die Darstellung des Zeitraumes, in dem das Objekt den abgebildeten Zustand zeigt

Granularität: Die ∼ ist der Verdichtungsgrad von↑ Fakten entlang der Ebenen der beteiligten ↑ Dimensionshierarchien

Hauptkomponentenanalyse: $\sim$ bezeichnet ein Verfahren zur $\uparrow$ Dimensionsreduktion, bei dem versucht wird, Regressoren aus der Regressionsgleichung zu entfernen, die nur wenig Erklärungszuwachs zu einer zu erklärenden Variablen liefern und demnach möglichst ohne Informationsverlust aus der Gleichung entfernt werden können

Header: Ein $\sim$ enthält Meta-Informationen, die bei jedem $\uparrow$ Request oder jeder $\uparrow$ Response mitgeschickt werden

Hierarchieebene: Die Daten in den Dimensionen lassen sich auswertungsorientiert zusammenfassen. Eine solche Zusammenfassung wird als $\sim$ bezeichnet

Histogramm: Ein $\sim$ ist eine Visualisierung von $\uparrow$ Binning-Methoden in Form eines Diagramms

Hit: jeder Zugriff auf einen Teil (ob Seite, Bild oder Text) eines Web-Angebots, der im Log-File des Servers eingetragen wird, wird als $\sim$ bezeichnet

HOLAP: Wenn $\uparrow$ Detaildaten in $\uparrow$ Relationen gespeichert werden und gewisse Verdichtungen multidimensional gehalten werden, wird eine Mischform, das hybride $\uparrow$ OLAP (also $\sim$) verwendet

Homonymfehler: $\sim$ bezeichnet die fälschliche Zusammenführung zweier Objekte oder $\uparrow$ Tupel, die unterschiedliche Entitäten beschreiben, zu einer neuen Entität

HTML-Tag: Ein $\sim$ ist ein Auszeichnungselement von $\uparrow$ HTML

HTTP, Hyper Text Transfer Protocol: $\sim$ ist ein Protokoll, welches für die Abwicklung fast aller Kommunikation im World Wide Web zuständig ist

Interpage Strukturdaten: $\sim$ ist die Bezeichnung der Struktur mehrerer Seiten untereinander. In der Regel ist damit die Struktur gemeint, die durch Verlinkung zwischen den einzelnen Seiten implizit gegeben ist

Interquartilsabstand: Der ∼ ist der Abstand zwischen unterem und oberen ↑ Quantil

ISP, Internet Service Provider: ∼ ist ein Unternehmen, der Zugang zum Internet ermöglicht. Dies kann über Modems, ISDN, DSL etc. geschehen. Solche Unternehmen bieten meistens auch noch weitere Dienstleistungen im Internetumfeld an

Item: ∼ (oder Literale) sind Elemente, deren Mengen die Werte der ↑ Sequenzen bilden. Ein ∼ kann z.B. ein Element eines Warenkorbs sein, der wiederum für eine ↑ Transaktion steht

Kategorische Daten: Als ∼ bezeichnet man ↑ Nominale und ↑ Ordinale Daten, also Namen ohne Wert oder Namen mit einer darauf definierten Ordnungsrelation

KDD-Prozess: Knowledge Discovery in Databases bezeichnet den nichttrivialen Prozess der Identifikation valider, neuartiger, potentiell nützlicher und klar verständlicher Muster in Daten. Der Prozess beinhaltet dabei Schritte von der ursprünglichen Datenauswahl bis hin zur Interpretation der gefundenen Muster und der folgenden Umsetzung

Kenngrösse: ↑ Fakt

Klasse: Eine ∼ besteht im Kontext der ↑ Datenanalyse aus einer Menge von Objekten, die alle eine bestimmte Gemeinsamkeit haben, beispielsweise die ∼ der kreditwürdigen Bankkunden innerhalb der Menge aller Bankkunden

Kennzahl: ↑ Fakt

Kennzahlen: Hier ist eine ↑ Kennzahl ein Begriff aus der Statistik. Er bezeichnet Werte, die berechnet werden, um Datenmengen charakterisieren zu können. Beispiele sind der ↑ Median, die ↑ Quantile oder die ↑ Varianz

Kennzahlenattribut: ↑ Kennzahl

Klassifikation: Bei der ∼ besteht die Aufgabe darin, betrachtete Objekte einer vorher bestimmten ↑ Klasse zuzuordnen. Die Zuordnung findet dabei aufgrund der Objektmerkmale und der Klassifikationseigenschaften statt

Klassifikationshierarchie: Eine ∼ bezüglich eines ↑ Pfades ist ein balancierter ↑ Baum. Seine Kanten repräsentieren die funktionalen Abhängigkeiten

Klassifikationsknoten: Ein ∼ ist die Verdichtungsstufe innerhalb einer ↑ Klassifikationshierarchie

Klassifikationsschema: Das ∼ einer Dimension ist eine halbgeordnete Menge von Klassifikationsstufen mit einem kleinsten Element

K-means: ∼ ist ein partitionierendes, globales Verfahren, das allen Elementen eine exakte Zuordnung zu Gruppen gibt. Dabei werden Clusterzentren zur Clusterbildung verwendet

Kontingenztabelle: Eine ∼ oder Kreuztabelle ist das Ergebnis aus der Zusammenführung mehrerer Tabellen zu einer einzelnen Tabelle, um Vergleiche zwischen den Daten aus beiden Quellen anstellen zu können. Es ist eine Aufstellung statistischer Auszählungen meist nominal oder ordinal-skalierter Variablen

Kontinuierliche Daten: ∼ nehmen Werte aus einem kontinuierlichen Wertebereich an. Ein für einen solchen sind die reellen Zahlen. In diesem Sinne kann man kontinuierlich mit lückenlos beschreiben

Konzepthierarchie: Die ∼ ist eine Hierachie abstrakter Konzepte, die im ↑ Data Mining benutzt wird, um viele einzelne Elemente, für die sich evtl. nur schlecht Regeln finden lassen, zu allgemeinere Elemente zu abstrahieren. Diese allgemeineren Elemente nennt man Konzepte

Künstliches Neuronales Netz: Ein ∼ ist ein Netz, in dem viele gleichartige Units (Neuronen) miteinander verschaltet werden. Einzelne Neuronen erfüllen einfache mathematische Funktionen, deren Ausgang wiederum anderen Neuronen als Eingangsdaten dienen. Solche Netze sind geeignet für vielfältige Aufgaben wie ↑ Regression,

Clusteranalysen oder die Mustererkennung

Ladekomponente: Um speicherungs- und auswertfähigen Daten im Data Warehouse-Kontext weiterzuleiten, ist eine Komponente notwendig, damit die analyseunabhängigen ↑ Detaildaten in die ↑ Basisdatenbank zu übertragen. Eine andere Komponente muss die analysespezifischen Daten aus der ↑ Basisdatenbank transferieren

Logfile: ↑ Webserver protokollieren jeden Zugriff auf ein Element der Seite in einer Protokolldatei, deren Format durch die Konfiguration des Servers bestimmt wird

Median: Der ∼ bezeichnet den Wert, der eine ↑ Stichprobe bzgl. einer Variable in zwei Hälften unterteilt; die untere Hälfte ist kleiner als der ∼ , die obere größer

ME/R- Modell: Das multidimensionale entity/relationsship- Modell ergänzt das ER-Modell um drei Elemente, um die multidimensionale Semantik darzustellen

Metadaten: ∼ sind ein Begriff zur Beschreibung jeder Form von Informationen über Daten. ∼ bieten die Möglichkeit, Informationen aus dem ↑ Data Warehouse zu gewinnen (Informationen über Daten aus der↑ Basisdatenbank und dem ↑ Data Warehouse, physische Speicherinformationen sowie Zugriffsrechte, Qualitätssicherung und Informationen über Data-Warehouse-Prozesse). Sie dienen sowohl als Informationslieferant als auch zur Steuerung des ↑ Data Warehouse Managers für die verschiedenen Prozesse

Metadatenmangager: Der ∼ steuert die Metadatenverwaltung. Er stellt eine Datenbankanwendung dar, der das Versions- und Konfigurationsmanagement, das Integrations-, die Zugriffs-, die Anfrage- und Navigationsmöglichkeiten der ↑ Metadaten anbietet. Ferner liefert er die Schnittstelle für Lese- und Schreibzugriffe auf das ↑ Repositorium

Missing values: ∼ bezeichnen Datenfelder, die keinen Wert beinhalten

Modellbildung: ∼ ist die vierte Phase im ↑ CRISP-DM; der Bereich, der allgemein als ↑ Data Mining bezeichnet wird, umfasst die

Auswahl und die Anwendung verschiedener Modellierungstechniken, um die in dem ↑ Business Understanding festgesetzten Data-Mining-Ziele zu erreichen

Monitor: Ein ∼ soll Datenmanipulationen aufdecken. Es gibt meist einen pro ↑ Datenquelle

Monitoring: Das ∼ ist die Voraussetzung für die Anpassung eines ↑ Data Warehouse an die aktuelle Nutzung

Multidimensionale Datenbank: Eine ∼ ist eine auf Grundlage des ↑ multidimensionalen Datenmodells aufgebaute Datenbank. Analog sind multidimensionale Datenbanksysteme und multidimensionale Datenbankmanagementsysteme definiert

Multidimensionales Datenmodell: Das ∼ ist ein Datenmodell, das die Modellierung von Daten zu Analysezwecken ermöglicht. Wesentliches Charakteristikum ist die Unterscheidung von Daten in ↑ Fakten und ↑ Dimensionen, sowie die Möglichkeit der Bildung von Hierarchien auf den Dimensionen

Multidimensionales Schema: Ein Schema, das mit den Mitteln eines ↑ multidimensionalen Datenmodells erstellt wurde, wird auch ∼ genannt

Multiple Hierarchie: ∼ bezeichnet eine Spezialform der Hierarchie, bei der auf eine ↑ Hierarchieebene alternativ mehrere folgen können

mUML: Die multidimensionale Unified Modeling Language ermöglicht als UML- Erweiterung die Erstellung eines konzeptionellen, ↑ multidimensionalen Schemata. Die multidimensionalen Sprachelemente und deren Semantik hat die ∼ von der Multidimensional Modeling Language (MML) erhalten. Grundlage für die ∼ sind UML-eigene Erweiterungsmöglichkeiten, die eine Anpassung ohne Veränderung des Metamodells ermöglichen

Noisy data: „Noise“ bezeichnet in diesem Kontext einen zufällig fehlerhaften Wert oder eine Abweichung in einer gemessenen Variable ↑ (Rauschen). Man spricht auch von verrauschten Daten

Nominale Daten: ∼ bezeichnet Daten, deren Wertebereich nicht numerisch ist, sondern Wörter oder Kategorien darstellen. Ein Beispiel hierfür ist die Einteilung in Blutgruppen

Normalisierung: ∼ ist ein Verfahren, durch das bestimmte Eigenschaften für die Daten wie z.B. Redundanzfreiheit oder die Abwesenheit von Update-Anomalien für einen Datenbestand erzielt werden können

Normalverteilung: ↑ Gaußverteilung

Numerische Daten: ∼ sind ↑ Diskrete und ↑ kontinuierliche Daten, also Daten, die aus Ziffern bestehen

Numerische Datenreduktion: ∼ bezeichnet die Verkleinerung des Datenbestandes durch Methoden wie z.B. Stichprobenziehungen, lineare ↑ Regression oder ↑ Clustering

Objekt-Historie: Hier werden nur ↑ Gültigkeitszeiten verwaltet. Es kann für jedes Objekt festgestellt werden, wann es in der modellierten Welt wahr war

OLAM: ↑ Online Analytical Mining

One-to-one Marketing: Unter ∼ oder auch Individual Marketing versteht man das auf spezielle Kunden oder Kundengruppen ausgerichtete Marketingverhalten, dass sich vom Massenmarketing abwendet

Online Analytical Mining: Im Zusammenhang mit dem ↑ Data Warehousing und dem ↑ Data Mining tritt das sogenannte ↑ Online Analytical Mining (OLAM) in Erscheinung, welches die auf Data Warehouse operierenden Techniken des ↑ Online Analytical Processing (OLAP) und des ↑ Data Mining integriert. ↑ OLAM bietet Möglichkeiten an, ein Mining auf verschiedenen Teilgruppen und Abstraktionsebenen, welche basierend auf ↑ Datenwürfeln mit OLAP-Methoden gebildet werden, durchzuführen

OLAP: Online Analytical Processing ist eine Methode um in großen Datenbeständen visuell aufbereitete, deskriptive ↑ Analysen zu ermögli-

chen. Bei ~ handelt es sich um eine Form der manuellen ↑ Datenanalyse mittels ↑ Generalisierung und ↑ Spezialisierung. Die notwendigen Daten werden oft im ↑ Data Warehouse (DW) bereitgestellt

OLTP: Online Transaction Processing beschreibt den Arbeitsprozess, der von den klassischen, operationellen, transaktionsorientierten Datenbankanwendungen verfolgt wird

Ordinale Daten: ~ bezeichnet Daten, deren Wertebereich aus Wörtern oder Kategorien besteht, die untereinander eine feste Ordnung haben

P3P, Platform for Privacy Preferences: ~ ist ein Projekt des ↑ W3C mit dem Ziel, dass User problemlos beurteilen können, welche Daten auf einer ↑ Website gesammelt werden. Ebenso sollen ↑ Website Betreiber die Möglichkeit erhalten, ihre User aufzuklären, damit diese wiederrum Vertrauen aufbauen und frei entscheiden können, ob sie mit der Sammlung der Daten einverstanden sind

Pfad: ~ Pfad bezeichnet eine vollgeordnete Teilmenge von Klassifikationsstufen eines ↑ Klassifikationsschemas

Port: Ein ~ ist ein „Kommunikationskanal" im Internet. Verschiedene Services benutzen auch unterschiedliche ↑ Ports, um Ihre Kommunikation abzuwickeln

Postprocessing: ~ ist der vierte Teilprozess des allgemeinen KDD-Modelles; Die Ergebnisse des ↑ Data Mining werden hier verarbeitet und bewertet und die gewählten Methoden als auch der gesamte bisherige Prozess werden kritisch betrachtet

Preisangabenverordnung (PAngV): Wer Endverbrauchern Waren oder Leistungen anbietet oder als Anbieter von Waren oder Leistungen gegenüber Letztverbrauchern unter Angabe von Preisen wirbt, hat die Preise anzugeben, die einschließlich der Umsatzsteuer und sonstiger Preisbestandteile unabhängig von einer Rabattgewährung zu zahlen sind (Endpreise). Auf die Bereitschaft, über den angegebenen Preis zu verhandeln, kann hingewiesen werden, soweit es der allgemeinen Verkehrsauffassung entspricht und Rechtsvorschriften nicht entgegenstehen

Preprocessing: ↑ Datenvorverarbeitung

Primärdaten: Im Forschungsumfeld sind unter ∼ solche Daten zu verstehen, die direkt bei den Untersuchungen entstanden sind und noch in keiner Weise verändert wurden

Proxy Server: ∼ ist ein Rechner, der Anfragen von anderen Rechnern entgegen nimmt und an das Ziel weiterleitet. Ein Proxy wird oft in Firmen eingesetzt um als ↑ Firewall zu dienen. Zusätzlich kann ein ∼ auch Seiten aus dem WWW zwischenspeichern und somit als ↑ Cache fungieren

Pruning-Strategien: Im Fall von komplexen und tiefgeschachtelten Entscheidungsbäumen, bei der die ↑ Klassifikation ungesehener Objekte häufig ungeeignet ist, verwenden mehrere ↑ Entscheidungsbaumverfahren ∼ . Ein in einem ersten Schritt konstruierter eventuell tiefverästelter ↑ Baum wird dabei durch das Herausschneiden von Unterbäumen reduziert, die nur einen geringen Beitrag zur ↑ Klassifikation leisten. Neben dem hier beschriebenen Postpruning wird auch das Prepruning angewandt, bei dem Unterbäume, die voraussichtlich wenig Beitrag zur ↑ Klassifikation leisten können, bei der Konstruktion des Gesamtbaumes berereits nicht mit einbezogen werden

Pull: ∼ ist das selbst bestimmte Heraussuchen von Informationen aus dem Web

Push: ∼ ist das ungefragte Erhalten von vorselektierten Daten aus dem Internet. Aktiv beteiligt an der Auswahl des Dateninhalts ist ein Nutzer nur bei der Auswahl seines Interessenprofils

Quadrierte euklidische Distanz: Die ∼ beschreibt die Summe der quadrierten Differenzen zwischen den Werten der Einträge

Quantil: Das ∼ ähnelt dem ↑ Median, nur daß die Grenze beim unteren ∼ nicht bei der Hälfte sondern bei einem Viertel liegt. Analog ist das obere ∼ definiert

Rauschen: Als ∼ bezeichnet man anormale Daten, die von ihrer Charakteristik her nicht in die Menge der Gesamtdaten passen

Recommender System: Ein ∼ schlägt dynamisch, anhand des aktuellen User-Verhaltens und aufgrund gesammelter Daten, alternative Links vor

Redundanz: ∼ bezeichnet die mehrfache Speicherung identischer ↑ Tupel oder Datensätze ohne Informationsgewinn

Referenzarchitektur: Die ∼ eines ↑ Data Warehouse System genügt den Prinzipien der Lokalität, der Modularisierung, Hierarchisierung sowie integrierter Dokumentation und Mehrfachvererbung

Regression: Mithilfe der ∼ versucht man eine Gleichung aufzustellen, die aus vorhandenen Attributen $(x_1 \ldots x_n)$ eine Variable y erklärt. Bei der klassischen linearen ∼ hat die Gleichung die Form $y = b_0 + b_1 x_1 + \ldots + b_n x_n + e$, wobei die b die zu wählenden Koeffizienten darstellen und e den sogenannten Fehlerterm. In der nichtlinearen ∼ sind auch bspw. exponentielle Gleichungen möglich

Relation: Die ∼ beschreibt die Teilmenge eines kartesischen Produktes $M_1 \times \ldots \times M_n$. Solche Relationen kann man als Tabellen z.B. in Datenbanken darstellen

Relationale Datenbanken: Kennzeichen relationaler Datenbanken ist, daß jede Information als Tabelle dargestellt werden kann und das alle Tabellen miteinander verlinkt werden können, insofern sie ein ↑ Attribut teilen

Repositorium: Das ∼ speichert die ↑ Metadaten des Data Warehouse-Systems

Request: Ein ∼ ist die Anfrage an einen ↑ Webserver nach dem ↑ HTTP-Protokoll

Response ∼ bezeichnet die Antwort eines ↑ Webservers auf einen ↑ Request

Rollback-Relation: In der ∼ wird die ↑ Transaktionszeit gespeichert. ↑ Transaktionen können so rückgängig gemacht werden

Roll-up: ~ bezeichnet das Zusammenfassen von ↑ Fakten aufgrund gleicher Ausprägung der qualifizierenden Eigenschaften, wobei die einzelnen quantifizierbaren Eigenschaften unter Verwendung von ↑ Aggregationsfunktionen zusammengeführt werden

Sampling: Wenn man statt alle Daten zu berücksichtigen, nur einen Teil berücksichtigt, spricht man von ~

Scatterplots: ~ werden zur graphischen Darstellung einer Variablenverteilung benutzt

Schnappschuss-Datenbank: ~ ist die Momentaufnahme einer ↑ temporalen bzw. einer konventionellen nicht-temporalen Datenbank

Secure Socket Layer (SSL): ~ ist eine Verschlüsselungstechnologie für die sichere Übermittlung von sicherheitsrelevanten Daten wie z.B. Kreditkartenangaben und Passwörtern

Self Organizing Map: ~ sind eine von Kohonen entwickelte Form der graphischen Clusteranalyse, die auf Technologien im Bereich der ↑ Künstlichen Neuronalen Netze zurückgehen. Ein SOM-Algorithmus kann selbständig Daten gruppieren und diese in einer zweidimensionalen Karte (Map) darstellen. Das Ziel von Self Organizing Maps (SOM) besteht in der topologieerhaltenden Abbildung hochdimensionaler Merkmalsräume in einen Outputraum niedriger Dimension. ~ sind dabei in der Lage, unbekannte Strukturen in der zu analysierenden Datenbasis ohne a priori Informationen selbstständig zu extrahieren und zu visualisieren

Sequenz: Bei einer ~ handelt es sich um eine Folge von Werten, die sich auf einander folgende Zeitpunkte oder Zeiträume bezieht

Slice und dice: ~ bezeichnet die benutzergesteuerte Erforschung eines Datenbestandes. Der Anwender kann während dieses Vorgangs Teile eines ↑ Datenwürfels selektieren, Datenwerte aggregieren oder transformieren, unterschiedliche ↑ Datenwürfel miteinander verknüpfen oder einen ↑ Würfel aus verschiedenen Perspektiven betrachten

Slowly changing dimension: Dieses von Kimball 1996 entwickelte Konzept beschäftigt sich mit den Werteänderungen der Dimensions-

attribute

Snowflake-Schema: Dieses Schema ermöglicht es, ↑ Klassifikationen direkt in einer relationalen Datenbank darzustellen, indem für jede Klassifikationsstufe eine eigene Tabelle angelegt wird. Durch die funktionalen Abhängigkeiten sind die Dimensionstabellen normalisiert. Dadurch werden ↑ Redundanzen reduziert (d.h. also Speicherplatz gespart) und Änderungsanomalien verhindert, allerdings kostet es viele Verbundoperationen

SOM: ↑ Self Organizing Maps

Spezialisierung: ~ ist die Der ↑ Generalisierung entgegengesetzte Methode, um zusammengefasste Daten wieder in Daten mit einem höheren Informationsgehalt umzuwandeln

SRSWOR, Simple Random Sample WithOut Replacement: ~ bezeichnet eine Stichprobenart: Es wird ohne Ersetzung ein Datum aus dem Datenbestand gezogen

SRSWR, Simple Random Sample With Replacement: ~ ist eine Stichprobenart: Es wird ein Datum aus dem Datenbestand gezogen und anschließend wieder dem Datenbestand hinzugefügt

Stammdaten: ~ sind Daten, die sich nicht oder nur selten ändern. Sie müssen nur einmal eingegeben werden und stehen dann ständig zur Verfügung

Star-Schema: Dieses Datenmodell bildet über die Faktentabelle und Dimensionstabellen die typischen ↑ OLAP Objekte ab: ↑ Datenwürfel, ↑ Dimensionen, ↑ Dimensionenhierarchien und Dimensionenelemente. Diese relationale Realisierung vermeidet das Entstehen teurer Verbundoperationen, indem die Tabellen einer ↑ Dimension zu einer einzigen Tabelle denormalisiert werden. Die Faktentabelle enthält die eigentlichen Analysedaten und ist weiterhin normalisiert, während die Dimensionstabellen, die nur beschreibende Daten beinhalten, dagegen verstoßen

Steam-and-Leaf-Plot: ~ bieten eine graphische Darstellung der Variablenverteilung

Streuverluste: ∼ sind durch Werbemaßnahmen erzielte Kontakte mit Personen, die nicht der definierten Zielgruppe zugehörig sind. ∼ werden auch Fehlstreuung genannt

Subsequenz: Eine ∼ ist eine Folge von Werten, die in einer anderen ↑ Sequenz enthalten ist. Dabei ist nicht die Übereinstimmung aller Werte entscheidend, sondern die Reihenfolge selbiger

Summierbarkeit: ∼ bezeichnet die inhaltliche Korrektheit der Anwendung einer ↑ Aggregationsfunktion auf einen ↑ Würfel

Supply chain: Wörtlich „Beschaffungskette". Eine ∼ ist ein Netz aus Einrichtungen, Distributionszentren und Verkaufsstellen, die erforderlich für den Materialfluss sind

Support ∼ ist die Häufigkeit, mit der ein ↑ frequent itemset oder eine ↑ Sequenz in der Datenmenge vorhanden ist. Machmal ist der ∼ auch als Quotient aus der Häufigkeit und der Gesamtgröße der Datenmenge definiert

Synonymfehler: ∼ bezeichnet das Nichterkennen der Zusammengehörigkeit zweier ↑ Tupel oder Objekte, die die selbe Entität beschreiben

Task Analysis: ∼ ist der erste Teilprozess des allgemeinen KDD-Modelles; in dieser Phase wird das Umfeld der ↑ Datenanalyse betrachtet und die Ziele für den weiteren Prozess festgesetzt

Temporale Datenbank: Eine ∼ ist eine Datenbank, die auf ↑ Gültigkeits- und ↑ Transaktionszeit basiert und es somit erlaubt, die vollständige „Geschichte" von Daten nachzuvollziehen

Transaktion: Eine ∼ bezeichnet den Vorgang des Speicherns in die Datenbank bzw. Vornehmen einer Änderung in der Datenbank. Einer Transaktion liegt das sogenannte ACID-Prinzip zu Grunde. Eine Transaktion muß somit den Prinzipien Atomicity, Consistency, Isolation, Durability genügen

Transaktionsdaten: Daten, die sich häufig ändern bzw. stets neu anfallen (z.B. Daten der laufenden Geschäftstätigkeit) nennt man ∼

Transaktionszeit: Die ∼ bezeichnet den Zeitraum zwischen dem Transaktionszeitanfang, an dem eine ↑ Transaktion in der Datenbank gestartet wird, und dem Transaktionszeitende, an dem sie abgeschlossen wird

Transformationskomponente: Die ∼ bringt die Daten, die in die ↑ Basisdatenbank geladen werden sollen, in einen geeigneten Zustand, da sie sich strukturell und inhaltlich unterscheiden

Tupel: ∼ ist ein Kunstwort, das zur Verallgemeinerung der Begriffe Paar, Tripel usw. benutzt wird. Ein n- ∼ ist eine aus n Elementen bestehende mathematische Größe. Im Zusammenhang mit einer ↑ Relation kann man ein ∼ als eine Zeile einer ↑ Relation verstehen

Tupel-Zeitstempelung: Die ↑ Zeitstempelung erfolgt hier pro ↑ Tupel. Dabei kann nicht pro ↑ Attribut unterschieden werden. Diese ↑ Zeitstempelung entspricht der ersten Normalform und ist mit konventionellen DBMS konform

URL Parameter: ∼ ist ein Parameter, der an eine ↑ URL angehängt wird. Dieser Parameter kann durch den ↑ Webserver oder eine serverseitige Programmiersprache ausgewertet werden

Valid Time: ↑ Gültigskeitszeit

Varianz: Die ∼ gibt ein Streuungsmaß an, das eine Aussage darüber treffen soll, wie weit im Durchschnitt jede Variable vom Mittelwert aller ↑ Stichproben abweicht

Verbundbildung: ↑ Clustering

Vorverarbeitungsschritte: Unter ∼ versteht man Operationen auf Daten, die vor der eigentlichen Datenverarbeitung durchgeführt werden und die die eigentliche Datenverarbeitung erleichtern sollen, indem sie beispielsweise die Daten bereinigen

W3C, World Wide Web Consortium: ~ ist der Name einer Organisation, die sich die Definition, Weiterentwicklung und Pflege von Standards für das Internet zum Ziel gesetzt hat. ↑ HTML, ↑ XML, RDF u.s.w. sind Beisipiele

Ward-Verfahren: Mit dem ~ werden zuerst Mittelwerte für jede Variable innerhalb einzelner ↑ Cluster berechnet. Anschließend wird für jeden Fall die ↑ Quadrierte Euklidische Distanz zu den Cluster-Mittelwerten berechnet. Diese Distanzen werden für alle Fälle summiert. Bei jedem Schritt sind die beiden zusammengeführten ↑ Cluster diejenigen, die die geringste Zunahme in der Gesamtsumme der quadrierten Distanzen innerhalb der Gruppen ergeben

Web Access Pattern: ~ ist ein häufig auftretender Zugriffspfade im ↑ Web Log

Web Content Mining: ~ bezeichnet ↑ Data Mining auf dem Inhalt von Webseiten

Web Data Mining: Das Anwenden von ↑ Data Mining Methoden auf den Daten des Internets wird auch ~ genannt

Web Log: Das ↑ Logfile eines ↑ Webservers, in dem alle Zugriffe auf Dateien des ↑ Webservers mitprotokolliert werden, wird ~ genannt

Webserver: Ein Programm, das Webseiten ausliefert (z.B. Apache), heißt ~

Web Structure Mining: ~ bezeichnet das ↑ Data Mining auf Strukturdaten des Internets

Web Usage Mining: ↑ Data Mining auf den Nutzungsdaten des Internets heißt ~

Würfel: Ein ~ besteht aus Datenzellen, welche eine oder mehrere ↑ Kenngrössen auf ↑ Detailebene beinhalten

Würfelinstanz: Die ~ ist eine Menge von Würfelzellen

Würfelschema: Das ∼ besteht aus der ↑ Granularität und einer Menge von ↑ Fakten

XML Extended Markup Language: ∼ ist eine Definition, um Strukturinformationen für Daten zu generieren

XML-Tag Ein ∼ ist ein Auszeichnungselement nach der XML-Definition

Zeitreihe: ↑ Sequenz

Zeitstempel: Ein ∼ ist eine Teilmenge der Zeit, über die ein Objekt definiert ist. Dieser kann entweder ein Ereignis definieren, ein Zeitintervall oder ein zeitliches Element

Zeitstempelung: ∼ unterteilt sich in ↑ Attribut- bzw. ↑ Tupelzeistempelung